AI4T – Advanced Intelligence for Testing

Von der Forschung in die Realität

nagarro

Gefördert von der Österreichischen Forschungsförderungsgesellschaft (FFG) hat ein Team von Automatisierungs- und KI-Experten an der Vision eines hochautomatisierten Test Life Cycle geforscht. Ihr Ziel: QA-bezogene Prozesse umfassender, skalierbarer, schneller und vor allem intelligenter zu machen.

Im Softwareengineering ist das Thema Softwaretest inzwischen fester Bestandteil der Entwicklungsprozesse. Auch die Automatisierung von Tests hat sich bewährt und trägt zu kürzeren und effizienteren Entwicklungszyklen bei. Viele Teams integrieren automatisierte Tests bereits in ihren Entwicklungspipelines und führen schon beim Einchecken von neuem oder geänderten Sourcecode, Testfälle auf mehreren Teststufen aus. Durch solche Maßnahmen fallen Fehler schneller auf und finden im besten Fall gar nicht erst den Weg ins gemeinsame Code Repository. Aus Sicht der Qualitätssicherung könnte man meinen, damit ein zufriedenstellendes Qualitätsniveau erreicht zu haben.

Aus der täglichen Praxis

Nicht bloß graue Theorie für Softwareentwickler

Die Realität sieht leider doch oft anders aus. Die aufwendig, und mühsam erkämpften Fortschritte bringen neue und vorher, oft nicht bedachte Herausforderungen mit sich, wie folgendes Beispiel-Szenario veranschaulicht:

Tom, unser UI Entwickler, hat die neuen Designs an unserem Log-In für unsere Web-Anwendung umgesetzt. Nach seinem Commit schlagen die Check-in-Tests fehl. Da Tom den Task vor Sprintende noch abschließen möchte, akzeptiert er den Umstand und checkt den Code ein – den dazugehörigen Test korrigiert er gleich morgen früh, denkt er sich. Am nächsten Morgen stellt Bernd, unser Backend Entwickler, fest, dass im Nightly Testrun ca. 40 Testfälle nicht erfolgreich durchgelaufen sind.

Mal eben schnell die Log Files analysieren...

Nach zwei Stunden zeitraubender Analyse der Log-Files, um den wirklichen Fehler herauszufinden und einem ernsthaften Gespräch mit Tom, glaubt Bernd alle relevanten Testfälle aktualisiert zu haben. Um sicher zu gehen, startet er die Regressions- und Systemtests.

Nun heißt es auf den Testdurchlauf warten...

Da bereits ein sehr umfangreiches Testset implementiert wurde, benötigt der Durchlauf der Tests 10 Stunden. Somit weiß Bernd erst am nächsten Tag, ob alles fehlerfrei funktioniert. Am darauffolgenden Morgen stellt er fest, dass einige der Tests über Nacht fehlgeschlagen sind. Die Analyse gestaltet sich kompliziert und dauert diesmal den ganzen Vormittag. Das Ergebnis ist eine bisher nicht entdeckte Fehlfunktion der Legacy Backend-Komponente.

Warum ist das Problem ausgerechnet im Legacy Code?

Das Problem betrifft jene Legacy-Komponente, die kaum dokumentiert und getestet ist, und deren Entwickler natürlich längst nicht mehr verfügbar sind. Man hätte die Komponente schon lange optimieren müssen. Dies wird nun eine sehr langwierige Aufgabe.

Nur die Tests selbst sind automatisiert

Die geschilderte Situation ist für die meisten Softwareentwickler nicht bloß graue Theorie. Die heute existierende Testautomatisierung hilft ohne Zweifel die Softwarequalität zu erhöhen. In unserer vorhin beschriebenen Geschichte wäre sonst der Fehler der Backend-Komponente nicht aufgefallen. Dennoch sind viele Prozesse rund um die Testautomatisierung immer noch manuell und aufwendig. Ziel ist es in Zukunft solche und ähnliche Herausforderungen zu meistern und den überwiegenden Teil des Test-Prozesses zu automatisieren.

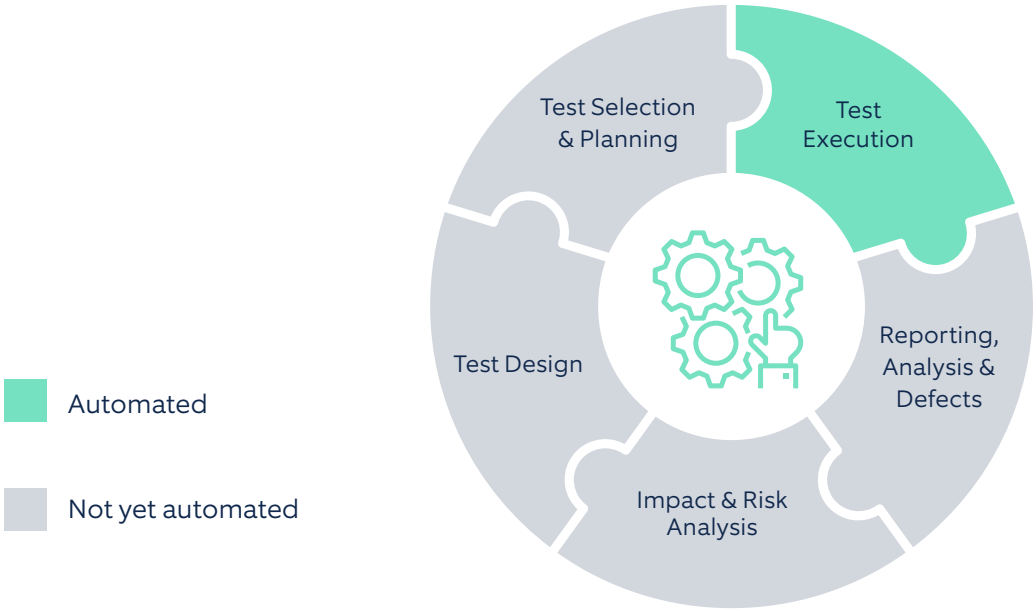

Abb. 1: Die automatische Testdurchführung ist lediglich ein kleiner Bestandteil im Test Life Cycle. Die verbleibenden Schritte sind meist noch manuell und wiederkehrende Aufgaben.

Die Vision: Ein hochautomatisierter Test Life Cycle

Gefördert von der Forschungsförderungsgesellschaft Österreich (FFG) hat das Technologieunternehmen Nagarro, das die globale Business Unit „Accelerated Quality and Test Engineering“ aus Österreich heraus steuert, gemeinsam mit dem Technikum Wien in 2019 ein Forschungsprojekt gestartet: AI4T - Advanced Intelligence for Testing. Durch den Einsatz von Machine Learning und Artificial Intelligence will man der Vision eines hochautomatisierten Test Life Cycles ein Stück näher kommen.

Bereits im ersten Jahr des Forschungsprojektes konnten vier Ansätze entwickelt werden, die Tom und Bernd, die Protagonisten in unserer Geschichte, sowie auch so manchen Tester und Softwareentwickler sehr gut in der täglichen Praxis unterstützt hätten. Im Folgenden stellen wir diese im Detail dar.

1. Smart Test Selection

Mit steigender Komplexität und Umfang von Anwendungen und Systemen wachsen die verschiedenen zu durchlaufenden automatisierten Testfälle schnell an. Ein vollständiger Testdurchlauf des kompletten Testsets benötigt meist einige Stunden und ist somit trotz Automatisierung für den untertägigen Test einer kleinen Anpassung zu zeitaufwendig.

War dies vor ein paar Jahren noch ein Luxusproblem, erleben wir heute derartige Situationen immer öfter. Häufig wird ein Subset dieser Tests als Smoke-Tests vor jedem Commit in den Master Branch durchgeführt. Die Herausforderung besteht dann darin, aus dem Pool der verfügbaren Tests die relevanten auszuwählen und dabei die richtige Balance zwischen Testabdeckung, Aussagekraft der Ergebnisse und Testdurchlaufzeit zu finden. Werden nur wenige Tests durchgeführt, besteht das Risiko Fehler zu übersehen. Dauert der Test zu lange führt dies zu unerwünschten Wartezeiten.

Laufzeit reduzieren und trotzdem alle relevanten Fehler finden?

Smart Test Selection beschäftigt sich mit genau diesem Problem. Es beantwortet die Frage, welche Teilmenge der Testfälle ausgeführt werden muss, so dass die Laufzeit reduziert wird, aber trotzdem möglichst alle relevanten Fehler gefunden werden.

Dabei werden mit AI-Methoden Charakteristika vergangener Testdurchläufe analysiert und aus diesen Daten eine Teilmenge der Testfälle ausgewählt. Wichtige Faktoren sind hierbei:

- Testfälle, die in der Vergangenheit häufig fehlgeschlagen sind, werden häufiger ausgewählt.
- Testfälle, die bereits lange nicht mehr ausgeführt wurden, werden wahrscheinlicher ausgewählt.
- Testfälle, die für gerade geänderte Codefragmente relevant sind, werden wahrscheinlicher ausgewählt.
- Testfälle, die eine lange Laufzeit haben, werden weniger wahrscheinlich ausgewählt.

Die ausgewählte Teilmenge der Testfälle muss im Zielbereich eine ähnlich vollständige Testabdeckung haben, wie die vollständige Testsuite. Andere Metriken wie das Alter des Testfalls, die Häufigkeit der Änderungen an dem Testfall, Aussagekraft des Testfalls und viele mehr können ebenfalls berücksichtigt werden.

Diese Faktoren werden mit Hilfe von intelligenten Modellen gewichtet und ein Subset der Testfälle ausgewählt. Mit diesem Framework können nun geeignete Selektionen von Testfällen sowohl für einen kurzen Smoketest, als auch für einen ausführlichen, aber dennoch zeitlich begrenzten Nightly-Run ausgewählt werden.

Bei diesem Vorgehen ist es wichtig, nicht durchgeführte Tests für spätere Testdurchläufe höher zu priorisieren. Dadurch lässt sich sicherstellen, dass Testfälle mit der Zeit nicht vom Radar verschwinden und keine Testlücken entstehen.

2. Self-Healing Test Automation

Ein vielfach unterschätztes Thema ist der Wartungs- und Pflegeaufwand, den automatisierte Testfälle mit sich bringen. Anpassungen am Code, aber auch Änderungen in der Laufzeitumgebung, an externen Schnittstellen oder an der Systemkonfiguration können Auswirkungen auf zuvor automatisierte Testfälle haben. So führt eine Umbenennung eines Buttons der GUI oder eine gut gemeinte Korrektur im Code schnell zu einem scheinbaren Fehler. Erst im Nachhinein stellt sich dann heraus, dass es sich um eine „False Positive"-Meldung handelt – und somit nicht um einen Fehler im Code, sondern um einen nicht mehr zum Code passenden Testfall.

Testfälle bei Änderungen automatisch im Quellcode anpassen.

Bei Self-Healing Test Automation beschäftigt man sich mit der Fragestellung, wie Testfälle bei Änderungen im Quellcode (semi-)automatisch angepasst werden können. Somit Warten und Pflegen sich die Testfälle automatisch und es wird der Aufwand der Anpassung der Testfälle stark reduziert.

Einfache UI Änderungen können oftmals sogar vollautomatisch durchgeführt werden. Bei komplexeren Änderungen ist es wichtig, dem Programmierer die Änderungen zu zeigen und bestätigen zu lassen.

Der Prozessablauf einer Self-Healing Lösung, die beispielsweise UI Komponenten korrigiert, sieht wie folgt aus:

- **Monitor:**
  Der Testdurchlauf wird überwacht und Daten über die Testfälle und UI Komponenten werden kontinuierlich gesammelt.
- **Detect:**
  Wenn ein Testfall fehlschlägt, wird dies sofort erkannt.
- **Analyse:**
  Nun startet die Analyse, ob die Ursache des Fehlers automatisch behoben werden kann. Hierbei wird mittels intelligenter Algorithmen ermittelt, ob der Fehler aufgrund eines verschobenen und/oder umbenannten UI Elements verursacht wurde.
- **Heal:**
  Wenn dies der Fall ist, werden die neuen Element Identifier automatisch in den Testfall übernommen und auch für zukünftige Durchläufe gespeichert.

Technologisch lässt sich dieses Problem mit AI basierten Methoden, wie zum Beispiel Graph Matching lösen. Hierbei wird der ursprüngliche UI Graph mit dem neuen UI Graph verglichen. Dabei wird anhand von Faktoren wie der Position, Name oder Textbeschreibungen das wahrscheinlichste UI Element im neuen Graphen selektiert.

Im Falle, dass mehrere ähnlich wahrscheinliche UI Komponenten in Frage kommen, wird der Nutzer benachrichtigt und kann von den verschiedenen Möglichkeiten auswählen.

3. Automatic Fail Analysis

Große Softwareapplikationen produzieren zahlreiche Log-Dateien mit einer großen Menge an Einträgen damit man im Fehlerfall den genauen Fehlerhergang nachvollziehen kann. Die Analyse dieser Log-Files im Fehlerfall ist nicht nur zeitaufwendig, sondern es ist oft auch schwierig den eigentlichen Fehlergrund herauszufinden, den Root-Cause.

Aus Log-Einträgen den wahren Fehler extrahieren

Viele Programmierer wenden einfache Heuristiken an, wie die Suche nach Begriffen wie ‚Error' oder ‚Exception'. Der wahre Grund des Fehlers ist häufig jedoch nicht direkt von der Fehlermeldung ablesbar, sondern versteckt sich in früheren Log-Einträgen. Wenn Testfälle fehlschlagen kann es gut sein, dass nur einige wenige Log-Einträge wirklich auf einen Fehler hindeuten und viele andere lediglich aufgrund von Folgefehlern passieren. Auch kann es sich um bereits bekannte Fehler handeln, die aufgrund der Weiterentwicklung nun erneut auftreten.

Die Automatic Fail Analysis beschäftigt sich mit Methoden, um aus Log-Einträgen den wahren Fehler zu extrahieren. Hierfür stehen zahlreiche Algorithmen und Verfahren im Bereich Machine Learning zur Verfügung.

Im Rahmen des Forschungsprojektes AI4T beschäftigt man sich mit dem Clustern von Log-Files. Hier ist die Idee, dass die Entwicklerteams sinnvolle Kategorien festlegen, in die Testdurchläufe anhand der produzierten Log-Einträge kategorisiert werden. Die Wahl der Kategorien hängt sehr stark von der Anwendung ab. Denkbar sind Einteilungen nach Bereichen der Anwendung, nach Layers (e.g. Datenbankfehler, UI-Fehler) oder nach Wichtigkeit des Fehlers.

Nach einer Zeit der manuellen Annotation durch Entwickler, werden diese Kategorien dann automatisch durch Machine Learning zugeordnet. Durch gezieltes Feedback der Entwickler zu den Vorschlägen, kann die AI dann mit jedem weiteren Testlauf dazulernen. Über die Zeit wird diese Zuordnung somit sukzessive verbessert.

4. Smart Refactoring

Code Qualität verbessern und Wartbarkeit des Codes erhöhen

Von der ersten Idee hin zu einer Anwendung oder einem System entstehen nicht selten Prototypen und Provisorien die, als Opfer ihres eigenen Erfolgs, ungewollt ihren Weg in das fertige Produkt finden. Der dabei entstandene unsaubere Code wird dann als „technische Schuld" durch die Jahre mitgetragen. Auch das Aufkommen neuer Technologien, Methodiken und Paradigmen erschwert den Umgang mit Legacycode. Workarounds, Adapter oder die nächste Middleware versprechen zwar schnelle und einfache Unterstützung, tragen aber zur ultimativen Fragmentierung bei, erhöhen die Komplexität des Gesamtsystems und reduzieren gleichzeitig dessen Wartbarkeit.

Smart Refactoring ist definiert als ein Prozess, der Softwarecode in einer Weise verändert, der die externe Funktionsweise nicht verändert, aber die interne Struktur verbessert. Durch diese Verbesserung der internen Struktur wird die Code-Qualität verbessert und somit die Wartbarkeit des Codes erhöht.

Das traditionelle Smart Refactorings analysiert den Sourcecode mit heuristischen Verfahren und wendet etablierte Design-Patterns an, um den Code umzustrukturieren. Machine Learning Verfahren können in diesem Prozess an vielen Stellen helfen.

Ein Ansatz ist zum Beispiel die Verwendung von Machine Learning, um auf der Basis von hunderten Open-Source Projekten zu lernen, wann solche Design Patterns tatsächlich angewendet und verbessert werden. Somit erhält man ein Machine Learning Modell, welches nur solche Änderungen vorschlägt, die in ähnlicher Weise auch tatsächlich von erfahrenen Programmierern in zahlreichen Open-Source Projekten umgesetzt wurden.

Erste Erfolge und vielversprechende Aussichten

“Wir werden den Menschen beim Testen niemals ersetzen können und dies ist auch gar nicht das Ziel von AI4T.”

Die vier beschriebenen AI4T-Use Cases lassen sich wunderbar auf unser exemplarisches Beispiel von Tom und Bernd umlegen: Bernd hätte durch intelligente Testauswahl keine 10 Stunden auf die Testausführungen warten müssen. Tom hätte es geholfen, wenn seine Umgestaltung der UI automatisch erkannt worden wäre und die Testfälle sich selbst an die neuen Komponenten angepasst hätten. Durch die automatische Fehleranalyse wäre sofort klar gewesen, dass es sich um einen UI-Fehler handelt und wer von den beiden sich drum kümmern muss. Und nicht zuletzt hätte Smart Refactoring vermeiden können, dass das Szenario mit einer Legacy Komponente endet, welche die Fehlerbehebung äußerst schwierig gestaltet wird. Der Nutzen von AI4T liegt klar auf der Hand: Zeitersparnis und Produktionssteigerung für Tom, Bernd und ihrem Team.

Die Ergebnisse aus dem Forschungsprojekt zeigen bereits nach dem ersten Jahr, dass viele Aspekte um den Test Life Cycle automatisiert werden können. Wir werden den Menschen beim Testen niemals ersetzen können und dies ist auch gar nicht das Ziel von AI4T. Aber es zeigt sich, dass durch den Einsatz von intelligenten Technologien viele von den ständig wiederkehrenden Aufgaben (teil-)automatisiert werden können. So bleibt mehr Zeit, um sich auf die komplexeren Fragestellungen zu fokussieren, die mehr Spaß machen!

nagarro

Über die Autoren

Jan Nössner ist Experte für Artificial Intelligence und Machine Learning bei Nagarro. Er leitete zahlreiche Machine Learning- und Datenintegrationsprojekte in verschiedensten Unternehmen.

Matthias Würthele ist Software Test Consultant bei Nagarro. Er bringt über acht Jahre Erfahrung in der IT-Projektleitung mit und hat zahlreiche Testprojekte erfolgreich begleitet.

Kontakt:
aqt@nagarro.com
www.nagarro.com/AI4T

About Nagarro
Nagarro begleitet Kunden in die Zukunft der Digitalisierung und zeichnet sich durch Unternehmergeist, ein agiles Mindset und das Wertesystem "CARING" aus. Gemäß dem Motto "Thinking Breakthroughs" überzeugt Nagarro durch die Entwicklung von digitalen Produkten auf höchstem Niveau. Über 8.400 Nagarrians in 25 Ländern unterstützen Unternehmen aller Branchen in der Transformation, um damit entscheidende Wettbewerbsvorteile zu erzielen.
www.nagarro.com