Test eingebetteter Systeme im industriellen Umfeld

nachgewiesene
Fehler ≙ Nichterfüllung einer Anforderung Istverhalten ≠ Sollverhalten
Ursache Fehlerzustand, daraus impliziert Fehlerwirkung
Ein Fehler kann einen anderen Fehler maskieren
Debugging soll Fehlerzustände beheben, Tests decken Fehlerwirkungen auf
Ein Test ist immer stichprobenartig
Testfälle → Testlauf, ~~Testläufe~~ → Testszenario
Testläufe korrellieren mit der Komplexität der Software
Statistisch: alle 200 Zeilen ein Programmierfehler
Spannungsdreieck Qualität / Kosten / Zeit bei nicht-lebenswichtigen Systemen sucht man den sweet-spot ~~X~~ Testen + Verbessern
Testautomatisierung spart Knete ↳ um die Qualität zu steigern
Die Methodik des Tests bestimmt die Effizienz des Tests.
Wie in SWT-2: Requirementsfehler zu beheben ist
~ 200x günstiger als wenn sie erst bei der Wartung, fertigem Deployment behoben werden. (FRONTLOADING!)
Qualitätssicherung:
Statisch / dynamisch (Prüfgegenstand wird ausgeführt nein/ja)
↳ formal / informell: Review erkennen semantische Korrektk.
Quellcode — Kompilat — Wissensaustausch unter Programmierern
Menschen sind visuelle Wesen, daher eignen sich grafische Modelle, um komplexe Sachverhalte abstrakt darzustellen,
Model Cheking

Funktionalität → Zuverlässigkeit → Effizienz → Benutzerfreundlichkeit
→ Wartbarkeit → Portabilität

Testmethoden

funktional: Lastenheft (Requirements) vs. Systemverhalten

Strukturell:

Statistisch: Operationales Modell oder komplett random

mutation. Test für Tests durch Fehlererzeugung im Prüfling

evolutionär:

Testorakel: kennt das erwartete Ergebnis "assertion"

Priorisierung: wichtigste Tests zuerst (falls die Zeit ausgeht

Metrik: oft genutzt, hohes Fehlerrisiko (Wahrscheinlichkeit * Kosten), Kundenerlebnis

Unabhängiges Testteam, das nicht programmiert hat

Zuverlässigkeitsanalysen für Software

Security (Sicherheit) gegen Knacker

Safety (Sicherheit) gegen Personenschaden

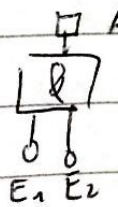
Dependability (Verlässlichkeit) für Verfügbarkeit

Reliability (Zuverlässigkeit) für Richtigkeit

MTBF / MTTF / MTTR

Echtzeitfähigkeit: hart / weich

Fehlerbaumanalyse: Gatter, Wahrscheinlichkeiten



Prognose für das zukünftige Verhalten auf Basis des Vergangenen

→ Herleitung der Parameter so, dass das damit modellierte Verhalten zur Probe/Vergangenheit plausibel ist.

λ ist keine zeitliche Komponente, wenn eine exponentielle Ausfallwahrscheinlichkeit angenommen

Testphase / Testprozess

Wasserfall: Lastenheft → Software requirement → Analyse →
Entwurf → Kodierung → Test → Abnahme

V-Modell eher bewährt ← lernen

Verschiedene Testmethoden:

Big-Bang: alles auf einmal

Bottom-Up: kleinste - nicht getestete Komponenten werden getestet

Top-Down: mit Stubs in tiefer Ebene

Collaboration: alles für einen Use-Case wird getestet

Feldtest: Einflüsse aus nicht vollständig bekannten Einsatzfeld einholen

Regressionstest: Keine neuen Fehler bei der Korrektur eines Anderen hinzugefügt

Testprozess lernen!

Statischer Test

keine Ausführung des Prüflings

kann alle Qualitätsmerkmale abdecken

Formales Review: Planung → Einführung → Vorbereitung
→ Reviewsitzung → Nachbereitung → Wiedervorlage

Planung: Dokumente sammeln

Einführung: Testobjekt verstehen

Vorbereitung: Ziele setzen

Sitzung: Mängel einstufen

Nachbereitung: Mängel beseitigen durch Änderung der Implementierung

Wiedervorlage: Folge-Review bei kritischen Fehlern

Informelles Review: nur Vor- und Nachbereitung, wird fast implizit verwendet bei Pair-Programming / Buddy Testing / Programmquelle wechseln

radix sort
♡

OC

Technische Reviews: mehr Aufwand bei der Vorbereitung
Inspektion: alles to the max.
Walkthrough: Schwerpunkt auf der Reviewsitzung, kaum Einführung und Vorbereitung

Statische Analyse

Prüfling muss formal vorliegen
Syntax check, Konventions-, Standardabweichung, Kontroll- oder Datenflussabweichung, Architektur und Designprüfungen
Dead Code, nicht verwendete Routine
Zyklomatische Komplexität: $e - v + 2$ (p)

Dynamischer Test

Funktionalen Test vs. Spezifikation
Unklares Test-Ende-Kriterium
Mind. 1 Vertreter jeder Test-Äquivalenzklasse wählen
Systematik beim Wählen der Stimuli (neg. Werte, Buchstaben, Grenzwerte)
Finden der relevanten Testaspekte
Kombinieren dieser CTE baut einen Tree aus Kombinationen
Zustandsbasierte Tests (Automat mit Zuständen)
Teste Alle Zustände (schwach), Alle Transitionen oder alle Ereignisse (stark, robust)

Strukturtest

Anweisungsüberdeckung
Zweigüberdeckung
Bedingungsüberdeckung
Mehrfach "

MCDC Modified Condition Decision Coverage
speckt die Mehrfachbedingungsüberdeckung um ihre Redundanz ab.

Segment-Pair-Überdeckung - Führe alle Teilpfade der Länge n zwischen Bedingungen aus

Def Use-Überdeckung - Teste alle Teilpfade zwischen Definition und Benutzung einer Variablen aus
p-use in lesenden Statements (if ...) ∀ enthält alle Zweige
c-use in rechnenden " (... = 4;)
all-use ∀ Def müssen alle p-uses und c-uses getestet werden
all-defs ∀ Def muss mindestens ein " " " " "

Pfadüberdeckung - teste alle Pfade von Beginn bis zum Ende des Graphen
findet Fehler, die bei bestimmten Zweigkombinationen auftreten (oder bei bestimmten Werten in Schleifen)

Back-to-Back Test - gegen eine andere Implementierung, kein Test gegen die Spec, sondern gegen . Dies kann ein Tool erledigen, da die Spec nicht gelesen werden muss.
Gemeinsame Fehler bleiben unentdeckt.
Skaliert nicht für umfangreiche Software

Mutationstest
Absichtliches Verändern der Software, um zu beurteilen, welche Testtechniken welche typischen Fehler entdecken können.
Starke Mutation erzwingt andere Ausgabe
Schwache " fordert nur ein anderes Verhalten

Regressionstest = ist, dass eine modifizierte Software keine neuen Fehler verursacht.

Statistischer Test

erzeuge zufällige Eingaben, die nach Operationsprofil gebeugt werden können. Auch hier Automatisierung möglich

Error Guessing

Basiert auf Erfahrungen von Programmierern

Höchste Effizienz?

Testumfang & Abdeckung

Metriken: Produktüberdeckung (Code, Daten, Zustände, Fehler)

Fortschritt (# Requirements abgedeckt

Aufwand (Dauer des Tests (* Manntage ⌀, # Fehler / Zeit

Historie (# Tests, # gefundene Fehler

Risiko (Abdeckung der Risikofaktoren)

Probleme (fehlende Testbarkeit, # Änderungen, fehlende Referenzen

Basis ausgewerteter Test (% Fehler in getesteten Bereichen)

Ergebnisse (Benchmarks, # Regressionstests, # offene Fehler)

Auswahl der Metriken: Vertrauen in den Code? Gesetz abdecken?

Wen adressiert die Metrik? Programmierer, Management, Kunden? Welche Instrumente können verwendet werden? Automatisierbar? Seiteneffekte? Zu starke Fokussierung auf ein Merkmal?

Ermittlung der Testabdeckung durch Tracing zu Requirements

Welche Req. wurden durch einen Fehler verletzt?

Was ist anzupassen & erneut auszuführen?

Alles abgedeckt?

Testfälle (Test Cases) haben Vorbedingungen, reihenfolge der Aktionen, ein erwartetes Ergebnis und ~~eine~~ Nachbedingungen (Prosa)

Testimplementierung

Händisch aus zu führende Liste von Aktionen

Testtool (PROVtech:TA, TPT, ...)

sind mit Testergebnissen verknüpft (obvsly.)

Test von Echtzeitsystemen

Kritikalität: Muss nicht besonders "schnell" sein!

Hart / Weich

Zum Test: Verbund, in dem das EES getestet wird, muss auch in Echtzeit laufen und die Stimuli müssen in ES gesetzt werden ⇒ Automatisierung, Reproduzierbar

MIL Model in the loop Software in the loop Processor II

Das EES muss beim Test die höchste Stimulations frequenz mindestens > doppelt so schnell abtasten. Nyquist-Frequenz

Auch Jitter müssen behandelt werden

Race Conditions: 2 Teilnehmen wollen gleichzeitig auf dieselbe Ressource zugreifen.

Deadlocks

Controller Area Network (CAN-Bus) kann ausgelastet sein, Nachrichten mit niedriger Priorität können verhungern (starving)

Anforderungen an die Echtteit-Tests:

alle nicht-Echtzeit-Anforderungen werden eingehalten

alle " " " "

Stresstest: Worst-Case Test für alle EES im SUT

Eingabedaten am Grenzbereich der Eingabe klassen (Wertebereichs)

Fehlerhafte / unplausible Eingaben, fehlende Eingaben

Timing an den zeitlich erlaubten Grenzen

Simulation von Jitter

Race Conditions erzeugen

Elektrische Fehlsteuerung, hochohmig, Kurzschluss, Unterspannung

Ausfall anderer Systeme

Erfahrungen aus anderen Tests einbringen

→ Umgebungssimulation notwendig

Zeitliche Genauigkeit

Schnittstelle & Pins

Benötigter Automatisierungsgrad

Fehlereinspeisung

Spezialllösungen für mechanische oder optische Schnittstellen

Software:

Umgebungssimulation (SimuLink)

Zeitliche Genauigkeit, parallele Vorgänge und automatisierte Auswertung vorhanden

Realzeittest mit realer HW Komponente

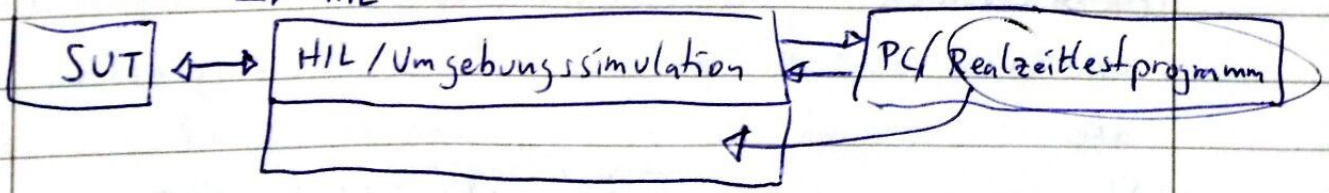
Daten-Player → SUT → Daten-Logger

Keine Simulation der Umgebung, (da keine Aktoren)

Schnittstellen meist größer als jene von

Wenig Automatisierung im Ablauf & Auswertung

=> HIL:



Simulationssprachen mit Echtzeitkomponente und Timer

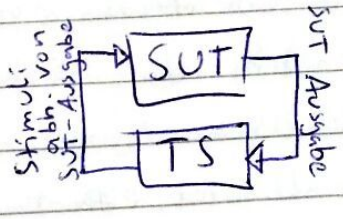
Modellbasierte Testverfahren

Lastenheft → Funktionsmodell → Tests
→ Testmodell → Target Code → Teststimuli → Output

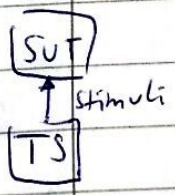
erlaubt parallele Entwicklung der Tests zur Implementierung
Zeitliche Komponenten wollen im Testmodell dargestellt werden
Nutzung Temporallogiken
LTL Syntax Xp, Fp, Gp, pUq, pRq
ermöglicht das Abfragen bestimmter Eigenschaften in einem gegebenen LTL Graph (Kripke-Struktur)
Markov-Ketten, W'keiten an den Transitionen, gedächtnislos
können auch wieder mit operationalen Profilen gebeugt werden

TPT (Time Partition Testing)

Ermöglicht das Testen von zeitkontinuierlichen Systemen
ES gemischt kontinuierlich und diskret in der Ein- und Ausgabe
Closed Loop



open Loop



E/E Testing

'Automatic Testing
Funktionsmodell, das alle Steuergeräte und Umgebungsvariablen beschreibt
Kann bestimmte, gewünschte Zustände erreichen und erzeugt dazu einen Test.

Hardware-in-the-Loop

Ein Steuergerät wird im Labor so verkabelt zu einer simulierten Umgebung, so dass es sich sowohl fühlt, als ob es sich im echten Fahrzeug verbaut wäre. Es merkt keinen Unterschied. Allerdings ist es möglich, die Stimuli gezielt zu simulieren und zu modifizieren.

SIL: Simulierte HW und Simulierte Umgebung
Rapid Prototyping: Simulierte HW & reale Umgebung
HIL: Reale HW & Simulierte Umgebung
Onboard Test: Reale HW & reale Umgebung

Grenzsituationen sind gezielt herbeiführbar
Automatisiertes Durchschreiten des Parameterraums
Reproduzierbarkeit von Tests
Alle Steuergeräte testbar in Echtzeit (1ms Zyklus) und sind miteinander & mit der Testsoftware verbunden
An allen Parametern schraubbar: Signale, Pegel, Lasten, Spannungen, Werten, Ströme, logische/elektrische Fehler
Was machbar ist: Funktionstest, CAN-Kommunikationstest, Netzwerkmanagement-Test, Powermanagement-Test, Diagnose-Test
Was nicht machbar ist: Elektromagnetische Verträglichkeit, Temperaturtest, Mechanische Belastbarkeit, Dauertest
Einzelnes System am Komponentprüfstand
Provetech: TA zum Schreiben von Scripts für den HIL-Simulator
Design 4 Testability: Steuergeräte so bauen, dass sie gut erreichbare Schnittstellen für Diagnosen haben.
Baureihen (un-)abhängige Bibliotheken für HIL Test,
Ziel: generisches Testprogramm bei versch. Bussen, Signalen ~.
=> Optimierter HIL-Betrieb

Evolutionärer Tests

Anwendung zur Pfadüberdeckung / Zweigüberdeckung / ...
" zum Testen funktionaler Eigenschaften
" zum Testen zeitlicher Komponenten

Definition einer Fitnessfunktion, Eingaben zum Test sind ein Individuum, Das Ergebnis des SUT wird bewertet

→ Finde jene Eingangsbelegung, die möglichst große Ausführungszeit erzeugt (finde WCET) ~ Seeding oder random

Verfahren: Initialisierung: Bewertung aller Individuen der Startpop
- Prüfe auf Erfolg / Abbruch
- Selektiere Beste Individuen
- Rekombination / Mutation / Einfügen

Fitnessfunktion relativ zu anderen Individuen aus der Generation (↦ Fortpflanzungswkeit)

Zielfunktion global, absolut bewertend

Rouletteselektion / Turnierselektion / Stochastic Universal Sampling / Truncation ↯ da einzige, bei dem die Diversität leidet, allerdings ist das Weiterbestehen des fittesten Individuums gegeben)

Rekombination: Erzeuge Nachkommen durch Aneinanderreihung exklusiver Bitstrings der Eltern.

Multi-Point-Crossover: Stücke haben zufällige Länge

Uniform-Crossover: Länge 1

Reduced-Surrogate-Crossover: Stücke beginnen immer dort, wo sich die Eltern Bits unterscheiden.

Mutation: Flippe eher das LSB als ein MSB

Addiere kleine zufällige Reele Zahl

Tausche zwei Variablen des gleichen Typs bei demselben Individuum a

Einfügen: Einfach: Nachkommen ersetzt Vorgängen 1:1

Zufällig: #Nachkommen < # Nächste Generation, Eltern

werden random eingefügt (bleiben erhalten)
Elitär: Bessere ersetzen Schlechte
Auswählend: #Nachkommen > #Generation, wähle Bestimme
Unterpopulationen: Mehrere autonome Evolutionen
Migration: Einfügen eines Individuums von einer anderen Unterpopulation, beschränkt Inzucht, erhöht Suchraum, erfahrungsgemäß schnelleres Finden guter Lösungen
~ Intervall bestimmt Häufigkeit der Migration
~ Rate bestimmt wie viele Migranten pro Migration
~ Topologie bestimmt wer wohin migrieren darf

Echtzeitsystem ist genau dann korrekt arbeitend, wenn zeitliche und logische Korrektheit vorliegt (auch zu früh kann schlecht sein, Airbags!)
Eignung für komplexe Systeme
— Bezogen auf Strukturtests:
Jeder Lauf durch das Programm ist ein Individuum
~~Redefinition~~ Redeklaration der Fitnessfunktionen: sollen nun bestimmen, wie gut die Variablenbelegungen durch verschiedene Zweige navigieren anhand des Abstands zu benötigten Zweigungskriterien.
— Bezogen auf Funktionstest
Idee: generiere die Eingaben für den Test so, dass die Funktion des Systems gebrochen wird
Individuen als Fahrmanöver
ET gibt Individuen dem HIL, der dann das Steuergerät testet

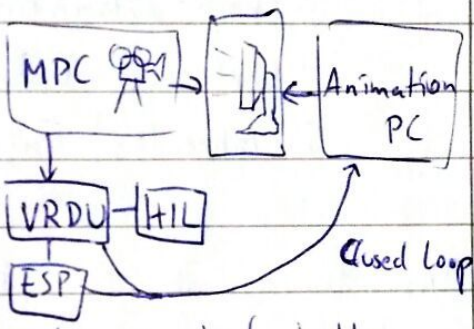
Einparkhilfe: Versuche Individuum zu finden, das beim Einparken ein anderes Fahrzeug oder den Bordstein (= Kollisionsbox) berührt

BAS (Brems-Assistenz-System): Finde Individuum, das unangemessen bremst, sei es unnötig (stark) oder zu schwach.

Fahrassistenzsysteme im Test

Für kamerabasierte FAS

Sensorik: Stereo-Kamera, Ultra-Schall, Radar (77-24 GHz), Laser alle → Sensor Fusion, um ein Bild der Umgebung zu schaffen, das dem des Fahrers im Kopf gleicht.



Sensorsimulation ⇔ kein physikalischer Sensor am Bus, dafür werden äquivalente Daten künstlich eingespielt.

Vehicle-in-the-loop

Reales, fahrendes Fahrzeug wird in ein simuliertes Umfeld gebracht.

Headtracker, HMD, halbtransparente Displays wegen Sonneneinstrahlung unpraktisch, non-transparent entfremdet Fahrer, Video-see-through erweitert VR-Bild um Innenraum & Fahrerkörper

Digitale Erprobungsfahrt

Testfahrszenario zufällig erzeugt, aber parametrisierbar

Zusätzlicher Input: Verkehrsmodell, Fahrermodell

Regelmodell prüft auf Kollisionen & sonstigen Verstößen

Simulationsmodell werden mit realen Daten gefüttert

Kontinuierliche Pass/Fail Kriterien

Absicherung autonomes Fahren

Stufe 0: Fahrer volle Kontrolle
1 (unterstützt): Füße weg
2 (teilautomatisiert): Hände weg
3 (hoch-automatisiert): Augen weg
4 (autonom): Gehirn weg
5 (fahrerlos): Fahrer weg

Anforderung an die Abnahme eines FAS ist deutlich niedriger, wenn für das FAS einen Fahrer als Rückfallebene gibt. Die Unfallszahlen der menschlichen Fahrweise sind zu unterbieten, ⇒ bessere Leistungsfähigkeit (Ø Strecke ohne Unfall)

Um eine signifikante Verbesserung zugrunde legen zu können und eine Täuschung (durch Zufall) ausschließen zu können, müssen 2,1 Mrd km Teststrecke zurückgelegt werden. → 36 Jahre mit 15k KFZ fahren ↝ nonsense

Dagegen zu tun: Wiederverwendung bereits freigegebenen Funktionen
evolutionärer Ansatz
Rafung der Testfälle, kritische Fälle zusammenbringen
Virtuelle Integration / Digitale Erprobungsfahrt
Intelligente Szenarien lernen, Runtime monitors
Erstelle positiv / negativ Szenario

Notwendigkeit: Erschaffung einer Metrik zur sinnvollen Freigabe für autonome Fahrzeuge

Werksführung nach Sindelfingen
~April

Virtuelle Integration

Validierung: modellbasierte Funktionsabsicherung
betrachtet: fahrdynamische Eigenschaften und Regelungen
Realitätsabstraktion, volles Fahrzeug & Umgebung virtuell
Ziel: schnelle Aussagen zu komplexem dynamischem Fahrverhalten
kein realer Test mehr ohne Echtzeit

Integration
Teile & Steuergeräte werden eventuell zu spät angeliefert,
schneller als Echtzeit
Nutze virtuelle Steuergeräte, diese Virtualisierung muss
umfassend existent sein, das schwächste, langsamste Glied
bestimmt die Gesamtgeschw.
Keine Möglichkeit auf Realkomponente zurückzugreifen, falls
Modellierung schwierig (bsp. Drosselklappe)
Mischbetrieb mit HiL möglich

Was wichtig ist: keine Formeln
Welche Testverfahren
Definitionen: Funktionstest? Methodisches (ÄK)
Klassifizierungsmatrix (Was ist HiL/SiL/Prototyp)
Evolutionären Test Erklären, Beispiele, Verfahren
Vor- und Nachteile (Komplexitätsraum...)
Zuverlässigkeitsmodell? → Aussagen gewinnen, Bsp.
Was macht man damit → Technologische Pfosten des HiL (↔ Parallelisierung)
und was nicht
Was ist der Loop beim HiL, warum nicht ohne?
Warum Echtzeit nicht
Sensorklassen bei FAS
das Maß allen Dinge ist.
Aufbau eines FAS im HiL
Was macht und ist TPT
Echtzeitbefehle von Skripten