Softwareprüfung gestern und heute: Theorie und Erfahrung, Standards und Common Sense

Prof. Dr.-Ing. habil. Peter Liggesmeyer
Lehrstuhl Software Engineering: Dependability
TU Kaiserslautern
Direktor Fraunhofer Institut für Experimentelles Software Engineering (IESE), Kaiserslautern

- Theorie
- Praxis
- Standards
- Erfahrung
- Common Sense

Theorie

- Testen:
  - Viele alte Techniken, die für viele Softwareentwickler immer wieder neu sind
  - Wenige grundsätzlich neue Theorien in den letzten 15-20 Jahren
  - Leistungsfähigkeit der Techniken schon theoretisch unklar (praktisch erst recht)
  - Theoretisch klar ist die Schwäche, dass Fehler “durchschlüpfen” können
  - Wird von vielen universitären Forschern als unergiebiges Arbeitsfeld gesehen => wenig Forschung, wenig Lehre dazu => siehe erster Punkt
- Formale Techniken:
  - Theoretisch ziemlich leistungsfähig, praktisch nicht
  - In der Praxis nur sehr eingeschränkt verwendbar
  - Theoretisch auch nur für sehr eng definierte Anwendungsgebiete

Theorie

- Analysetechniken:
  - Datenflussanomalienanalysen, Checken von Konventionen usw. theoretisch einfach
  - Viele Werkzeuge
  - Erstaunliche Ignoranz in der Praxis
- Quintessenz Theorie:
  - Testen ist theoretisch schwer zugänglich
  - Formales Verifizieren ist theoretisch interesssant, solange stark idealisierte Annahmen über die Rahmenbedingungen gemacht werden
  - Analysieren ist theoretisch trivial

Praxis
Notwendige, minimale Anforderungen an Tests

- Absolut notwendig entspr. aller maßgeblichen Standards:
  - Funktionsorientierte Testplanung für alle Testphasen
  - Reproduzierbarkeit von Testergebnissen => automatische Regressionstests nach Software-Modifikationen
- Weitgehender Konsens:
  - Ergänzende strukturorientierte Abdeckung (mindestens Zweigüberdeckungstest)
  - In kritischen Anwendungsbereichen – z. B. der Avionik – werden darüber hinaus durch Standards explizit gründlichere strukturorientierte Tests gefordert
  - Durchführung möglichst in der ersten Testphase nach Fertigstellung des Codes (Modultest)
  - Zusätzlich Leistungs- und Stresstests insbesondere in technischen Anwendungsbereichen

Praxis
Eine einfache praxisgeeignete Teststrategie

- Modultest
  - Funktionsorientierter Test der Module unter Verwendung eines Zweigüberdeckungstestwerkzeugs
    - Funktionsorientierte Testfallerzeugung (z.B. funktionale Äquivalenzklassenbildung)
    - Vorbereitung - d.h. Instrumentierung - der zu testenden Module zur Kontrolle der erreichten Zweigüberdeckung
    - Vollständige Durchführung der funktionsorientierten Tests
    - Kontrolle der auf diese Weise erreichten Zweigüberdeckung (erfahrungsgemäß ca. 70% - 80%)
  - Strukturorientierter Test der Module unter Verwendung des Zweigüberdeckungstestwerkzeugs
    - Ursachenanalyse für die Nichtausführung von Zweigen
    - Erzeugung von Testfällen für die noch nicht durchlaufenen Zweige
- Integrations- und Systemtest
  - Funktionsorientierter Test

Standards
Bedeutung von Standards

- Standards entscheiden im Zweifelsfall, welche Verfahrensweisen, Methoden und Techniken als Stand der Technik bzw. als Stand von Wissenschaft und Technik zu betrachten sind.
- Standards und Normen:
  - Keine Rechtsnorm , aber antizipierte Sachverständigengutachten
- Gesetzliche Regelungen:
  - z.B. Produkthaftungsgesetz, Schadensersatz nach BGB
- Europäische Richtlinien
  - Haben den Charakter eines Gesetzes, weil sie von den Mitgliedsstaaten zwingend in nationales Recht umzusetzen sind
- Verordnungen
  - werden meistens von Behörden – der Exekutive – erlassen und sind in der Regel verbindlich

Standards
Bedeutung von Standards

- Normung ist in Deutschland die planmäßige, durch die interessierten Kreise gemeinschaftlich durchgeführte Vereinheitlichung von materiellen und immateriellen Gegenständen zum Nutzen der Allgemeinheit. Deutsche Normen werden in einem privatrechtlichen Verein durch interessierte Kreise erstellt (z. B. DIN Deutsches Institut für Normung e.V., Verband Deutscher Elektrotechniker (VDE) e.V.). Standards und Normen sind keine Rechtsnormen. Sie sind – im Unterschied zu Gesetzen – nicht rechtsverbindlich, aber sie können als antizipierte Sachverständigengutachten verstanden werden. Durch Einhaltung der jeweils relevanten Normen kann ein Hersteller sicherstellen, dass der Stand der Technik erreicht ist, und er damit seine Sorgfaltspflicht erfüllt hat.

Prozessorientierte Standards

- Regeln z. B. die Verfahrensweisen, Abläufe, Aufgaben und Verantwortlichkeiten in der Software-Entwicklung und Software-Qualitätssicherung.
- Im Wesentlichen enthalten sie organisatorische Forderungen.
- Sie schließen kaum genaue technische Forderungen ein.
- Beispiele:
  - DIN ISO 9000-Reihe
  - V-Modell
  - ISO/IEC 15504 zum Assessment-Verfahren SPICE
  - AQAP-Century-Standards für den militärischen Anwendungsbereich

Technische Standards

- Technische Standards können entweder einen bestimmten Anwendungsbereich betreffen – z. B. Luftfahrt oder Schienenverkehr – oder auf bestimmte Arten von Systemen anwendbar sein, die in einer Vielzahl von Anwendungsbereichen auftreten können.
- Enthalten oft explizite Regelungen der einzusetzenden Techniken
- Beispiele:
  - IEC 61508 /IEC 61508 98/ ist ein sehr umfassender Standard zum Thema Sicherheit elektrisch bzw. elektronisch programmierbarer, sicherheitskritischer Systeme. Software wird insbesondere in der IEC 61508-3 behandelt.
  - Der Standard DIN EN 50128 /DIN EN 50128 01/ ist für die Anwendung auf Schienenverkehrssysteme vorgesehen.
  - Der Standard /RTCA/DO-178B 92/ betrifft Software-Anforderungen im Bereich der Avionik.

Standards und Softwareprüfung

- Alle Standards betonen die Bedeutung des funktionorientierten Testens
- Viele Standards enthalten explizite Regelungen der einzusetzenden Techniken
  - DIN EN 50128 erzwingt die Nutzung von Kodierkonventionen für C
  - RTCA DO-178 B regelt explizit die einzusetzenden strukturorientierten Testtechniken in Abhängigkeit der Kritikalität

Erfahrung

	/Fenton, Ohlsson 00/	/Basili, et al. 96/	/Cartwright, Shepperd 00/	/Basili, Perricone 84/	/Abreu, Melo 96/
Wenige Module enthalten die Mehrzahl der Fehler.	++	++	(+)	++	/
Wenige Module erzeugen die meisten Ausfälle.	++	/	/	/	/
Viele Fehler im Modultest bedeuten viele Fehler im Systemtest.	+	/	/	/	/
Viele Fehler im Test bedeuten viele Ausfälle im Feld.	--	/	/	/	/
Fehlerdichten korrespondierender Phasen sind über Releases hinweg konstant.	+	/	/	/	/
Umfangsmaße sind geeignet zur Fehlerprognose.	+	/	+	-	/

++: starke Bestätigung; +: schwache Bestätigung; 0: keine Aussage;
-: schwache Ablehnung; --: starke Ablehnung; /: nicht evaluiert;
?: unklar

Erfahrung
Erkenntnisse I

- Fehler sind über die Module einer Software nicht gleichmäßig verteilt, sondern konzentrieren sich in einigen wenigen Modulen
- Diese Module erzeugen den größten Teil der Probleme
- Großer Modulumfang bedeutet nicht zugleich hoher Fehlergehalt
- Viele erkannte Probleme im Test bedeuten nicht, dass eine Software in der Praxis Qualitätsmängel zeigt
- Es scheint Regeln zu geben, die dafür sorgen, dass aufeinander folgende Entwicklungen ähnliche Ergebnisse liefern

Frage:

- **Wie können die wenigen besonders fehlerhaften Module erkannt werden?**

Erfahrung

	/Fenton, Ohlsson 00/	/Basili, et al. 96/	/Cartwright, Shepperd 00/	/Basili, Perricone 84/	/Abreu, Melo 96/
Code-Komplexitätsmaße sind geeignete Mittel zur Fehler-prognose.	besser als Umfangs-maße: -	WMC: +	WMC: /	besser als Umfangs-maße: -	MHF: +
		DIT: ++	DIT: ++		AHF: 0
		RFC: ++	RFC: /		MIF: +
		NOC: ?	NOC: ?		AIF: (+)
		CBO: ++	CBO: /		POF: +
		LCOM: 0	LCOM: /		COF: ++

- Objektorientierte Maße:
  - WMC (*Weighted Methods per Class*)
  - DIT (*Depth of Inheritance Tree*)
  - NOC (*Number Of Children*)
  - CBO (*Coupling Between Object-classes*)
  - RFC (*Response For a Class*)
  - LCOM (*Lack of Cohesion on Methods*)
  - MHF: *Method Hiding Factor*
  - AHF: *Attribute Hiding Factor*
  - MIF: *Method Inheritance Factor*
  - AIF: *Attribute Inheritance Factor*
  - POF: *Polymorphism Factor*
  - COF: *Coupling Factor*

Erkenntnisse II

- Einzelne einfache Komplexitätsmaße (z.B. McCabe zyklomatische Zahl) funktionieren nicht besser als Umfangsmaße (z.B. LOC)
- Spezifischere Komplexitätsmaße zeigen oft eine gute Prognosequalität für den Fehlergehalt

Schlussfolgerung:

- **Eine geeignete Kombination von geeigneten Komplexitätmaßen gestattet eine gezielte Identifikation der fehlerhaften Module**

Erfahrung

	/Fenton, Ohlsson 00/	/Basili, et al. 96/	/Cartwright, Shepperd 00/	/Basili, Perricone 84/	/Abreu, Melo 96/
Modellbasierte *(Shlaer-Mellor)* Maße sind geeignet zur Fehlerprognose.	/	/	Events: ++	/	/
Modellbasierte Maße sind geeignet zur Prognose des Codeumfangs.	/	/	States: ++	/	/

Erkenntnisse III

- Aus Softwareentwürfen können Messwerte gewonnen werden, die es gestatten, Codeumfänge und Fehlergehalte frühzeitig zu prognostizieren

Common Sense zur Softwareprüfung

- Wir haben eine beachtliche Diskrepanz zwischen Theorie und Praxis
- Standards unterstreichen die praktische Bedeutung von Softwareprüfungen unter Nichtbeachtung der mäßigen Theorie
- Eine Aufarbeitung der theoretischen Seite ist nicht in Sicht
- Die praktische Bedeutung wird eher zunehmen
- Man muss heute zwingend die potentiell relevanten Standards kennen
- Es gibt einige interessante empirische Erkenntnisse, die man nicht ignorieren sollte => viel Unsinn aus der Vergangenheit ist heute widerlegt

Literatur

- /Abreu, Melo 96/
  Abreu F., Melo W., *Evaluating the Impact of Object-Oriented Design on Software Quality*, Proc. Metrics '96, pp. 90 - 99
- /Basili et al. 96/
  Basili V., Briand L.C., Melo W.L., *A Validation of Object-Oriented Design Metrics as Quality Indicators*, IEEE Transactions on Software Engineering, Vol. 22, No. 10, October 1996, pp. 751-761
- /Basili, Perricone 84/
  Basili V.R., Perricone B.T., *Software Errors and Complexity: An Empirical Investigation*, Communications of the ACM, Vol. 27, No. 1, January 1984, pp. 42-52
- /Cartwright, Shepperd 00/
  Cartwright M., Shepperd M., *An Empirical Investigation of an Object-Oriented Software System*, IEEE Transactions on Software Engineering, Vol. 26, No. 8, August 2000, pp. 768-796
- /Fenton, Ohlsson 00/
  Fenton N., Ohlsson N., *Quantitative Analysis of Faults and Failures in a Complex Software System*, IEEE Transactions on Software Engineering, Vol. 26, No. 8, August 2000, pp. 797-814