professional it software & services pitss®

Oracle Forms von A - Z

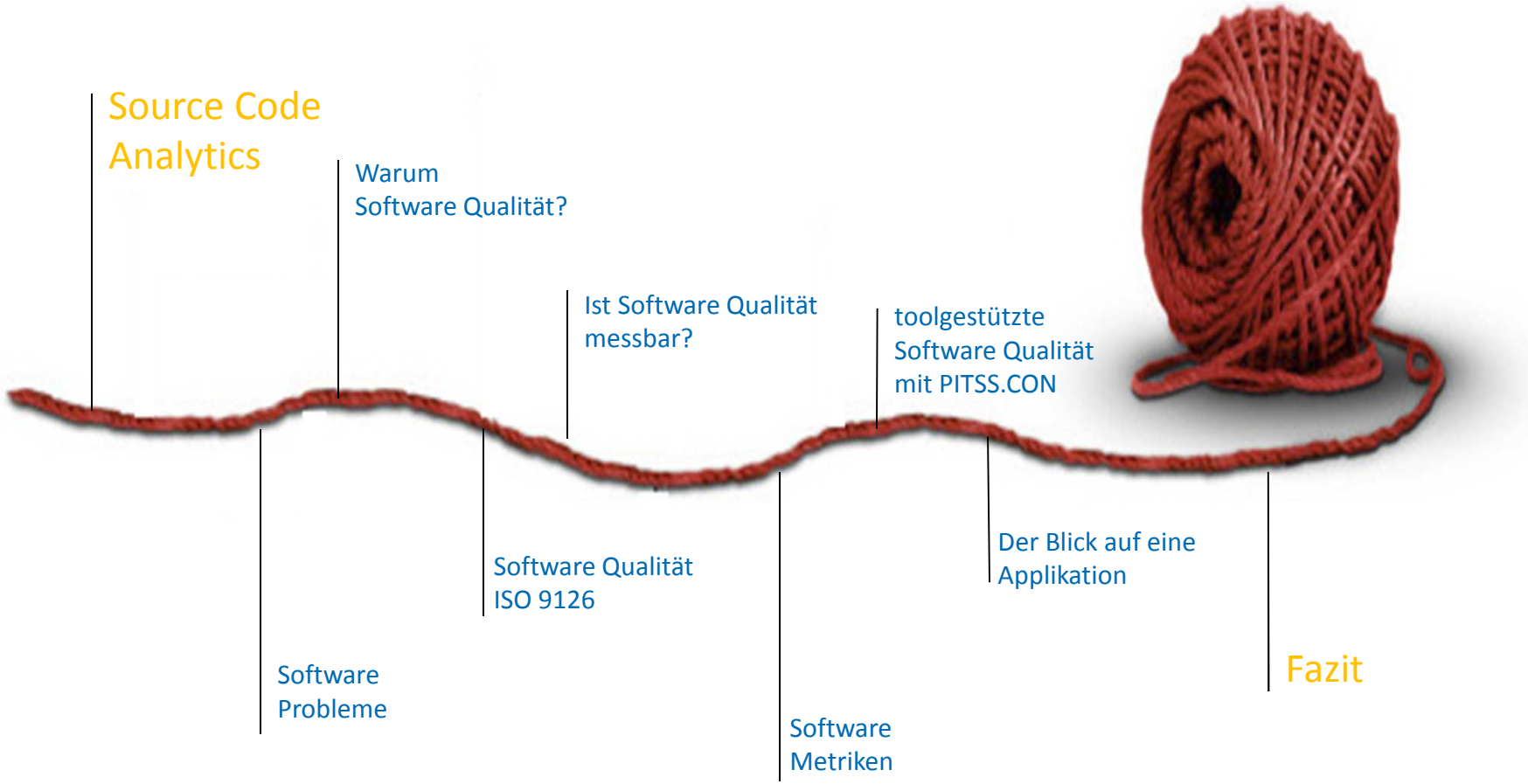
Source Code Analytics

Andreas Gaede

The Oracle Modernization Experts

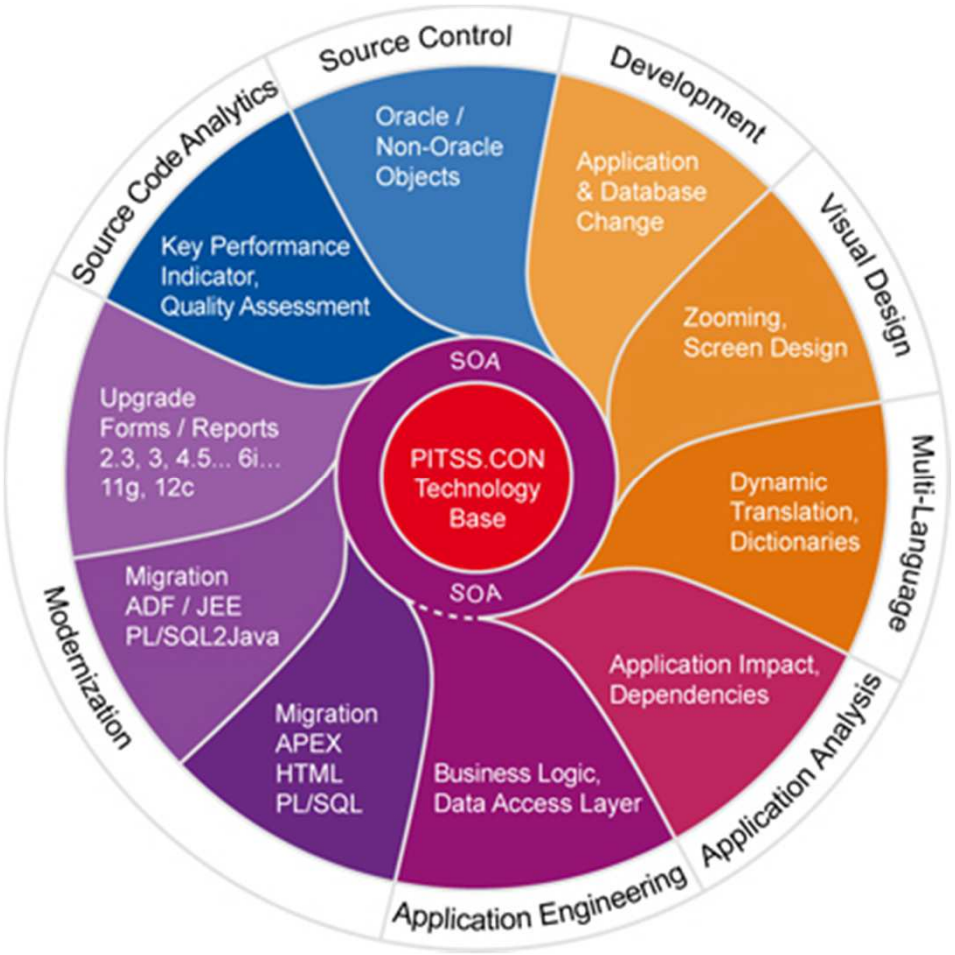
ORACLE Gold Partner

Agenda



PITSS.CON - Source Code Analytics

- Technology Base
- Maintenance / Development
- Graphical Visual Design
- Dynamic Multi-Language
- Application Analysis
- Application Engineering for SOA
- ADF- / APEX- Assistant
- Automatic Forms upgrading
- Source Code Analytics
- Source Control



Ariane 5

Am 4. Juni 1996 startete die ESA eine unbemannte Rakete mit vier Satelliten an Bord von Französisch Guyana aus. 40 Sekunden nach dem Start explodierte die Ariane 5.
Verlust ca. 500 Millionen Dollar für Rakete und Satelliten. Entwicklungskosten ca. 7 Milliarden Dollar.

Ursache für den Absturz:
Der Bordcomputer stürzte 36.7 Sek. nach dem Start ab als er versuchte, den Wert der horizont. Geschwindigkeit von 64 Bit Gleitkommadarstellung in 16 Bit signed Integer umzuwandeln: -+ b1 b2 ... b15.

Bemerkungen:
(1) Die Software stammte von der Ariane 4, aber die Ariane 5 flog schneller!
(2) Die Software war für den eigentlichen Flug überflüssig
und diente nur den Startvorbereitungen. Um einen möglichen Restart im Falle einer kurzen Unterbrechung des Countdowns zu ermöglichen, blieb das Programm 40 Sek. lang während des Flugs aktiv.
(3) Der Backup-Rechner verwendete exakt das gleiche Programm.
(4) Die Umwandlung war nicht abgesichert, da man glaubte, dass die Zahl nie so groß sein könnte.

Berühmte Softwarefehler

Das Denver-Koffer-Debakel

Bei der Neueröffnung des Flughafens in Denver sollte ein voll automatisches Gepäcksystem verwendet werden.
300 Computer, Laserscanner, Photozellen, Ethernet-Netzwerk.

Eröffnung des Flughafens wegen Fehler im Gepäcksystem um 16 Monate verspätet.
Zerquetschte, verlorene Koffer usw. Wieder Gepäck-Sortierung per Hand nötig.

Verlust: ca. 3.2 Milliarden Dollar.

Teilursache:
Zu viele Nachrichten über Ethernet.
Sortier-Anweisungen kamen nicht rechtzeitig
wegen Überlastung des Netzwerks (LAN).

Gesamtproblem zu komplex.

Warum Software-Qualität

Nach Untersuchungen von Standish Group, Gartner Group, Cutter Consortium und Center for Project Management:

- *23 % aller Softwareprojekte erfolgreich,*
- *53 % über Budget und/oder über Zeit und*
- *24 % abgebrochen*

International Data Corporation (IDC) - Umfrage

Umfrage zur Sicherung der Softwarequalität versus durch Softwaredefekte auftretenden internen und externen Kosten

Etwa 14 Millionen Euro (22 Millionen Dollar) Kosten bringen Unternehmen je nach Größe pro Jahr für die Fehlerbeseitigung auf

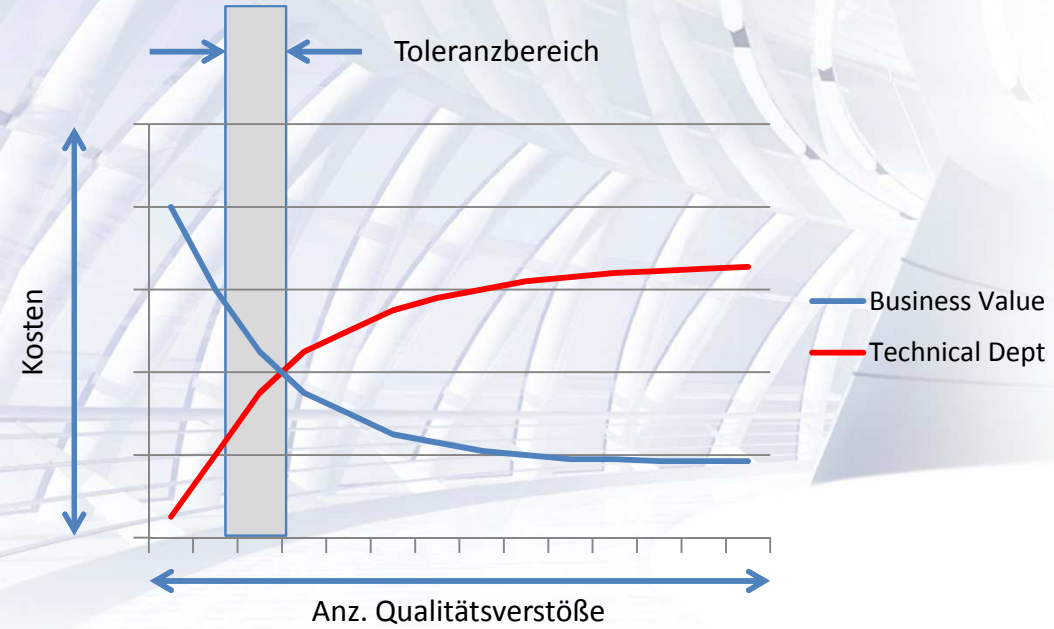
IDC-Umfrage sieht die Probleme mit der Softwarequalität in mehreren Faktoren begründet:

- wachsende Komplexität von Code,
- auf verschiedene Standorte aufgeteilte Teammitglieder,
- Outsourcing,
- veralteter Code,
- **Rückgriff auf Open-Source-Code** und
- das vermehrte Aufkommen von Multi-Thread-Anwendungen.

Allgemeine Betrachtung der Software-Qualität

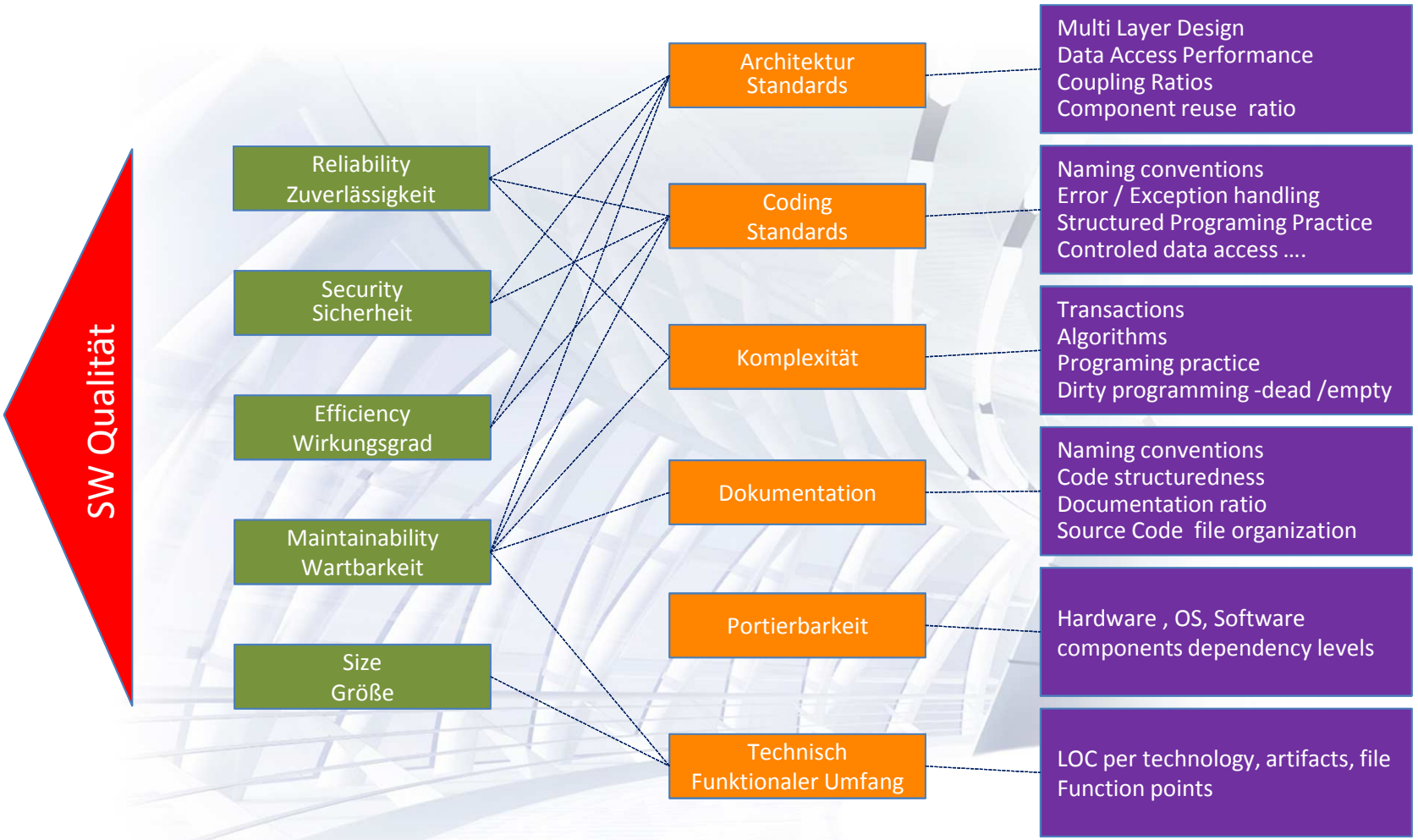
Gartner Group, Studie

- *374.000 Zeilen Programmcode enthalten ein mittleres finanzielles Risiko von kalkuliert 1Millionen $*
- *500 Milliarden $ beträgt diese „Technical Debt“ technische Hypothek weltweit*
- *1 Billionen $ das Doppelte im Jahre 2015*

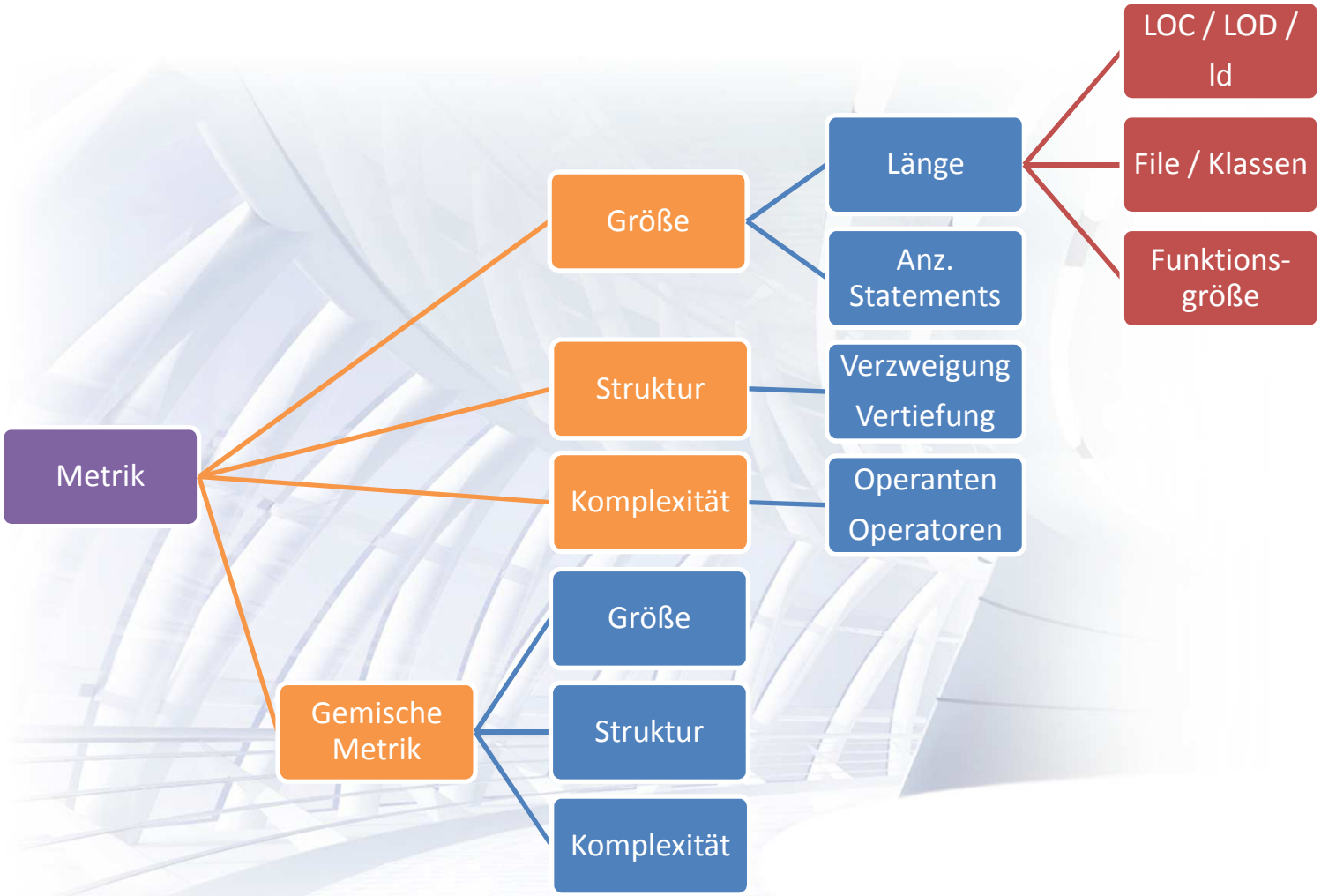


Software Qualität nach ISO 9126 (neu 25010)

wünschenswerte Charakteristik und messbare Eigenschaften



Metriken zur Messung der Software Qualität



Anzahl Statements – einfache Metrik

Gezählt werden „functional Operations“ (enden mit ;) im Main-Block ohne Deklarationen.

Rating	Risiko
0 – 100	small: Code-Zerlegung nicht notwendig
101 -250	Medium: möglicher Kandidaten zu Code-Zerlegung
> 250	Large: Kandidat zu Code-Zerlegung

Zeilen Metrik

LOC (*Lines of Code)* ist die gebräuchlichste Messung für die Quantifizierung der Komplexität einer Software.

+ Metrik ist einfach, leicht zu messen und sehr verständlich
- lässt die *Intelligenz* und den *Aufbau des Codes* unberücksichtigt

LOCphy: Anzahl der physikalischen Zeilen (Number of physical lines);
LOCpro: Anzahl der Programmzeilen (Number of program lines): Deklarationen, Definitionen, Direktiven und Code;
LOCcom: Anzahl der Zeilen mit Kommentaren (Number of commented lines);
LOCbl: Anzahl der Leerzeilen (blank lines), Hinweis: Leerzeilen innerhalb eines Kommentarblocks werden als Kommentarzeile gezählt.

	akzeptable Werte
4 – 40	LOCpro pro Programm / Funktion
10 – 100	Programme / Funktionen in einer Datei (Datei <= 400 LOCpro)
30 – 40%	Kommentare - gut dokumentiert
> 75%	ist keine Datei mehr sondern ein Dokument

McCabe Metrik (zyklomatische Komplexität)

Die zyklomatische Komplexität dient der Beurteilung der Programmstruktur und steht im Zusammenhang zum Programmieraufwand, Fehlerbehebung und Wartungsaufwand

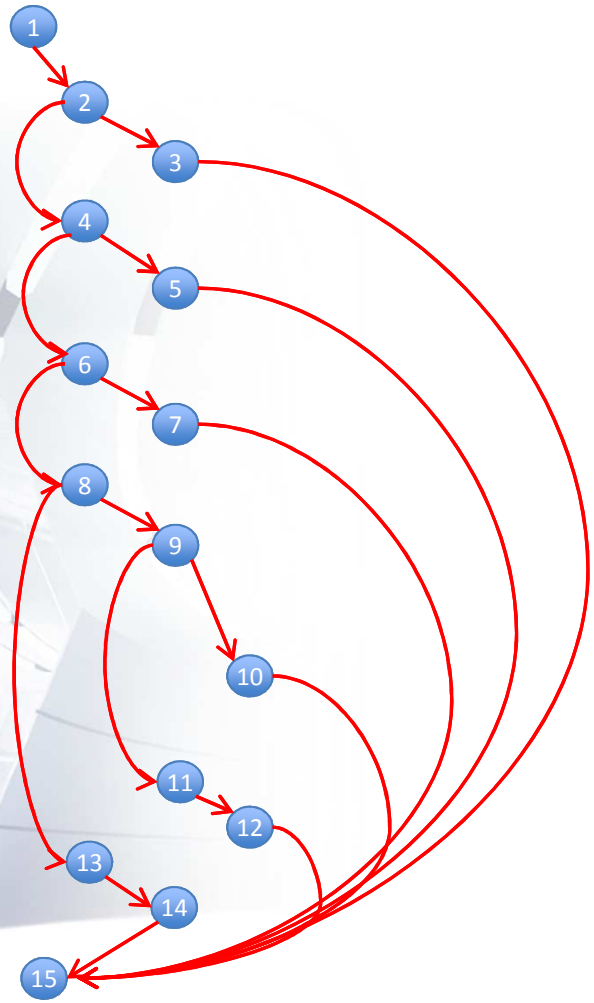
Komplexität nach McCabe $M = E - N + 2P$

M = zyklomatische Komplexität
E = Anzahl der Kanten (**E**dges) eines Graphen
N = Anzahl der Knoten (**N**odes) eines Graphen
P = Anzahl der verknüpften Komponenten (**P**rograms)

Rating M	Risiko
1-10	einfaches Programm, mit geringem Risiko
11-20	geringe Komplexität, akzeptierbares Risiko
21-50	komplex, hohes Risiko
➢ 50	kaum beherrschbares Programm, sehr hohes Risiko

McCabe Metrik (zyklomatische Komplexität)

```
void output( int auswahl, int detail)
{
        if (auswahl==1)
                printf("\n 1");
        else
        if (auswahl ==2)
                printf("\n 2");
        else
        if (auswahl ==3)
                printf("\n 3");
        else
        if (auswahl ==4)
                if (detail == 0)
                {
                        printf("\n 4");
                        printf("\n 5");
                }
                else
                        printf("\n 6");
        else
                printf("\n 7");
}
```



$$M = E - N + 2P \qquad E = 19\ N = 15\ P = 1 \qquad M = 19 - 15 + 2 * 1 = 6$$

Halstead Metrik

Basismaße für jedes Programm:

Anzahl der verwendeten unterschiedlichen Operatoren ($n1$) und Operanden ($n2$),
Wortschatz $n = n1 + n2$

Anzahl der insgesamt verwendeten Operatoren ($N1$) und Operanden ($N2$),
Implementierungslänge $N = N1 + N2$

Operatoren		Operanten
▪ if ▪ then ▪ else ▪ elsif ▪ exit ▪ case ▪ Close ▪ fetch ▪ when ▪ loop	▪ for-loop ▪ while-loop ▪ exit-when ▪ goto ▪ return ▪ open ▪ open-for ▪ open-for-using ▪ pragma ▪ exception	▪ Identifier (z.B. number ***test***) ▪ Numbers ▪ Characters ('x') ▪ Strings ("xyz") ▪ Exceptions
procedure call function call BEGIN END () [] (Each pair counts as one operator.)	AND OR NOT <> <= >= = != ; , : = . LIKE BETWEEN - + * / % < >	

Halstead Metrik

Halstead-Länge $HL = n1\ \log_2 n1\ + n2\ \log_2 n2$

Halstead-Volumen $HV = N\ \log_2 n$

Aus den Basisgrößen kann man verschiedene Kennzahlen berechnen:

Schwierigkeit (Difficulty): $D = \frac{n1}{2} * \frac{N2}{n2}$

Aufwand (Effort): $E = D * V$

Implementierungszeit (Time): $T = \frac{E}{18}\ Sekunden$

Bugs (ausgeliefert): $B = E^{2/3}/3000$

Rating HV	Risiko
0 – 1000	verständliches Programm, leicht zu warten
1001 – 3000	Herausforderung, akzeptierbares Risiko, Senior Skills
➢ 3000	zu komplex, Kandidat für Re-Design oder Re-Factoring

Halstead: gültig für alle Programmiersprachen, lässt sich automatisch ermitteln
Nachteil, nur einzelne Funktionen und lexikalische/textuelle Komplexität

Maintainability Index oder MI - Metrik

Ziel: Eine Mischung aus Halstead und McCabe sowie LOC und CM zu schaffen, die die Wartbarkeit besser beurteilt.

MI = 171 - 5.2 * ln(aveV) - 0.23 * aveV(g') - 16.2 * ln (aveLOC) + 50 * sin (sqrt(2.4 * perCM))

aveV = durchschnittliches Halstead Volume V pro Modul
aveV(g') = durchschnittliche Cyclomatic Complexity pro Modul
aveLOC = durchschnittliche LOC pro Modul
perCM = durchschnittliche % LOD pro Modul CM(Comment per Module)

Diese Metrik erlaubt eine bessere Einschätzung der Wartbarkeit, da ein kompliziertes Programm, das in der Halstead Metrik schlechte Werte bekommt, auf Grund einer ausgezeichneten, leserlichen Dokumentation einen guten Wert in der Wartbarkeit erreicht.

Rating MI	Risiko
< 64	schlechte Wartbarkeit (kann auch negativ werden)
65 -85	akzeptable Wartbarkeit
> 85	hohe Wartbarkeit

PITSS.CON und Software-Qualität

- Source Code Betrachtung für die gesamte Applikation
  - Programm und Programm-Typ übergreifend
- Code Parser Zerlegung der Programme auf kleinste Komponenten (Variablen, Deklarationen, Statements, Join-Conditions, Tabellen , Tabellenfelder, …..)
- gezielte Suche beliebiger Objekte in der Applikation
- Objektverfolgung (Call-Stack-Up und Call-Stack-Down)

➔ vielfältige Dokumentationen wo verwendet, wie verwendet
Online Dokumentation – mit Drill-Down und Drill-Up Funktionen

➔ Unused Code (identifiziert nicht verwendete Komponenten)

➔ ideale Umgebung, da die Größen und Kennzahlen für Software Metriken vorhanden sind

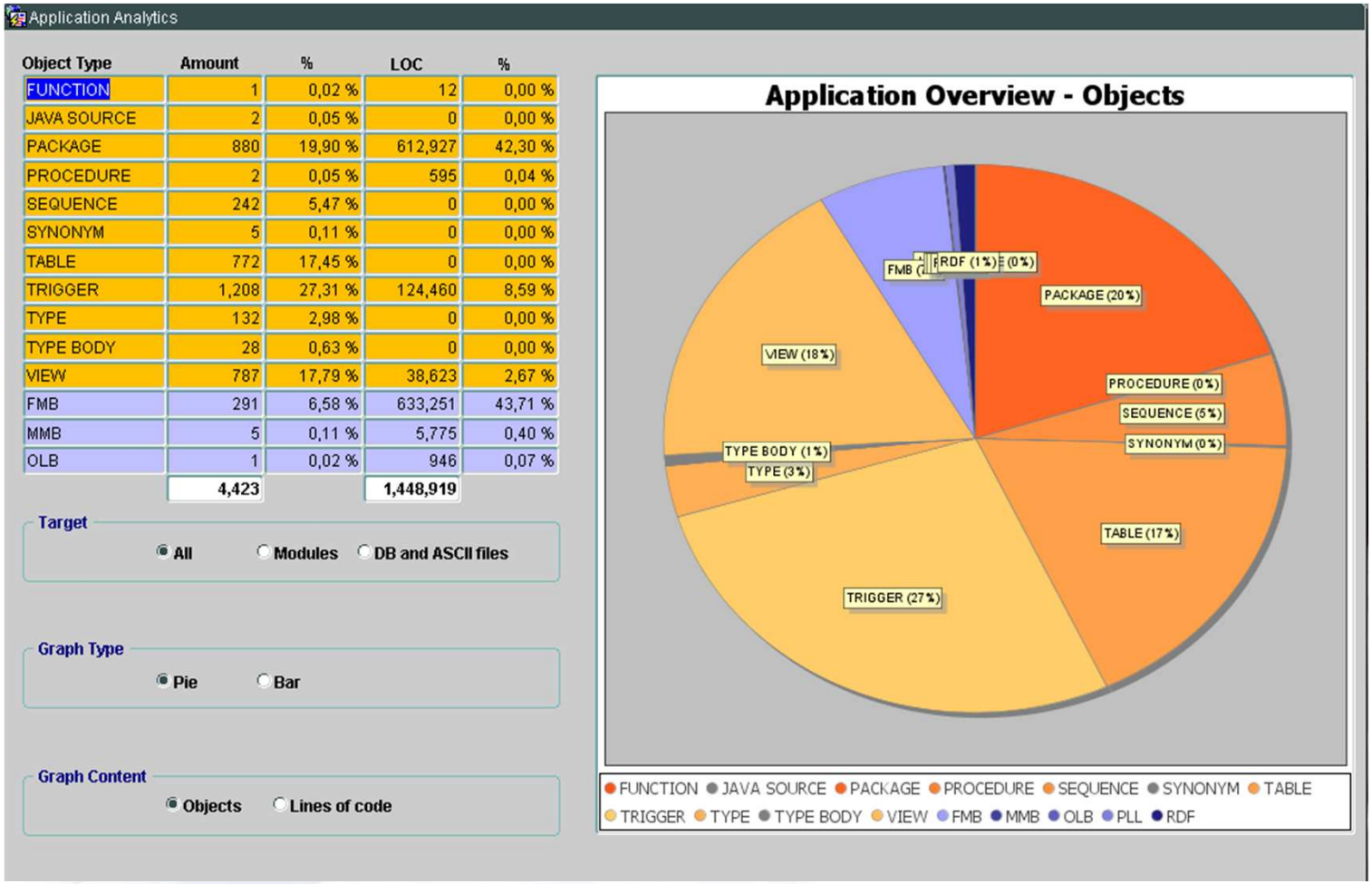
Source Code Analytics

Namenskonventionen	(Prüfung von Objektbezeichnungen)
Application Overview	Mengengerüst (Anz Programme, usw .)
Code Overview	LOC / LOD / ld (level of documentation)
DML Statement Overview	CRUD (Create/Read/Update/Delete) Verteilung der Statements
Function Overview	Mengengerüst von Funktions- / Prozedur-Aufrufe

Source Code Analytics

Operational Analytics	**Top Listen**, Objektbeziehungen, Codebearbeitung auf Datei-Typen und Programm-Typen
▪ Source Code	Programme nach LOC / LOD / ld Programm-Verwendung, -Metrik, CRUD
▪ DB-Operations	Datenbankoperationen CRUD / Programm Programm-Verwendung, -Metrik
▪ Tabellen / Views	in der Verwendung und Häufigkeit
▪ Program Units	in der Verwendung und Häufigkeit
▪ Software Metrik	Sortierung nach Metrik-Typ und Verwendung, Gesamt-Sicht

Live



Fazit

Sinnvolle Maßnahmen zur Schaffung einer besseren SW Qualität

- saubere, beherrschbare Architektur (Templates)
- Entwicklungsrichtlinien (Coding Standards)
  - Naming Conventions
  - Funktionszugriff (Access Control)
- Komplexität beachten / reduzieren
  - Programm- / Funktionsgröße
  - Struktur und Strukturtiefe
  - kleine beherrschbare Funktionen / Dateien
- Dokumentation
  - Spezifikationen
  - Comment Ratio (ld = level of documentation)
  - Test-Dokumentation
- Code Review
  - kontinuierliche Prüfung (*besonders bei Out-Sourcing*)

Fazit

Software Qualität macht Sinn

- verringert Fehler
- macht eine Applikation beherrschbar
  - Modularität / Portabilität / Lesbarkeit / …
- spart Aufwände in der Wartung und Weiterentwicklung
- erhält und steigert den Wert einer Applikation

Einsatz von Werkzeugen zur Absicherung der Software Qualität

- ein Muss bei größeren Anwendung (manuell kaum machbar)
- klare Ansätze
- Prüfungen wiederholbar
- beschleunigt und gewährleistet Qualitätssicherung

Fragen und Antworten

professional it software & services pitss®

Thank You

For Your Attention!

Andreas Gaede

The Oracle Modernization Experts

ORACLE Gold Partner