Künstliche Intelligenz in der Legacy Softwareentwicklung

Akademische Initiative

Uwe Graf, Senior Software Architect

EasiRun

Die Software-Modernisierer

Wer sind wir?

EasiRun Europa GmbH

Firmensitz Usingen (HE, Hochtaunuskreis)

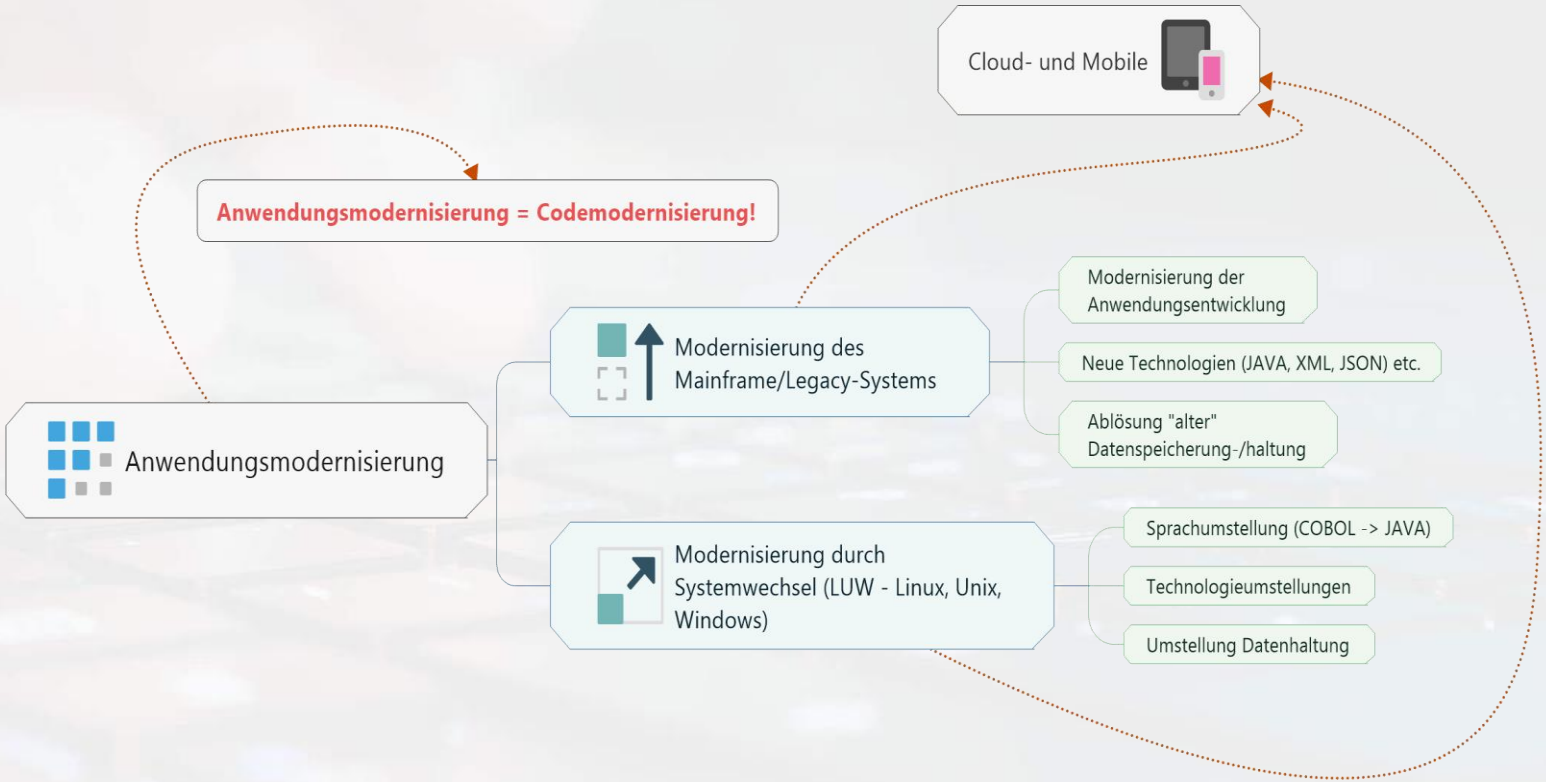
1992 gegründet als Tochterunternehmen der EasiRun International, **seit 1997 selbstständige GmbH**

Europaweit 70 Spezialisten, davon 30 in Deutschland (Festangestellte und Freelancer)

Geschäftsführer Donald Fitzgerald

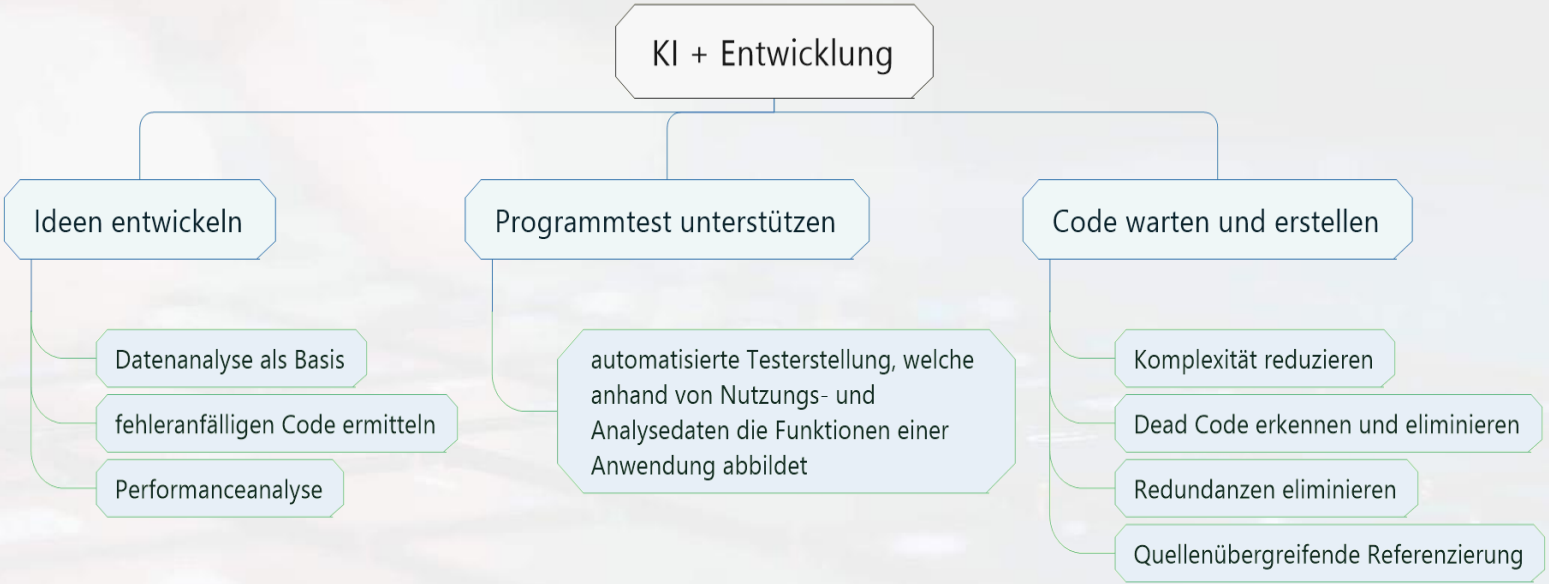
- Anwendungen
  - COBOL Compiler, Technologien
  - Integration JAVA,.NET,COBOL,PL/1
- Tools zur Anwendungsmodernisierung
  - Moderne IDE's für z/Systems
  - COBOL/JAVA Crosscompiler
  - Enterprise Software Analytics
- Services/Dienstleistungen
  - Enterprise Software Analytics
  - Anwendungsmodernisierung
  - Business IT Automation
- Cloud und Mobile für z/Systems

Modernisierung – der Ausgangspunkt!

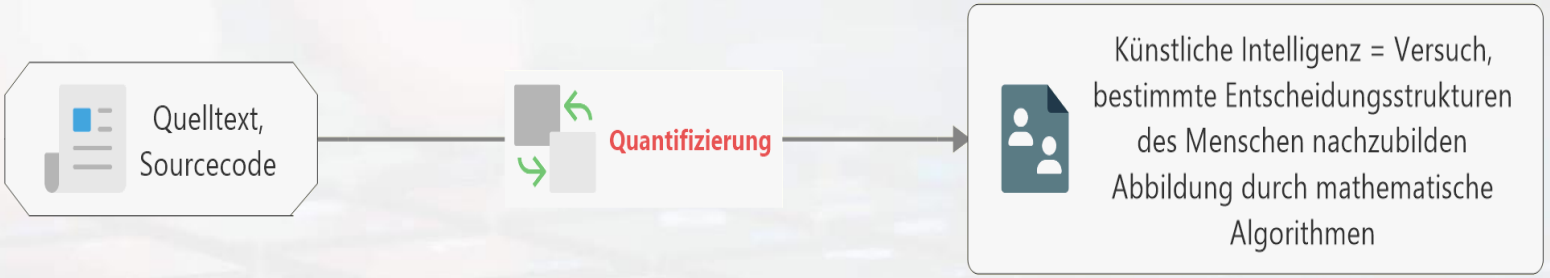


KI – Wie einsetzen?

KI – Wo einsetzen?



Quantifizierung als „Hilfsmittel“



Code Metriken als Basis der Interpretation (1)

- Metrik wird angewendet auf „Softwareeinheit“
- i.d.R. Softwareeinheit gleichgesetzt mit Quellcode
- Metrik repräsentiert funktionalen Zusammenhang oder wird aus Checkliste abgeleitet
- Einfache Metrik zeigt Größe des Quellcodes, komplexe Metriken versuchen die Verständlichkeit des Quellcodes zu bewerten
- Beurteilungsgrundlage bei Entwicklung, Weiterentwicklung
- Softwaremetriken können nicht die korrekte Umsetzung bewerten, sondern nur vorherbestimmen, welchen Aufwand die Erstellung der Software bereiten wird und wie viele Fehler auftreten können.

Code Metriken als Basis der Interpretation (2)

- Bekannte Metriken:
  - Anzahl der Codezeilen (Zeilenmetriken), engl. Lines Of Code (LOC)
  - das Function-Point-Verfahren zur Aufwandsabschätzung in der Analysephase
  - COCOMO zur Errechnung von Projektkosten aus anderen Kennzahlen
  - die zyklomatische Komplexität (nach McCabe) zur Komplexitätsbestimmung eines Programmodules
  - die Halstead-Metrik zur Implementierungsabschätzung in der Entwurfsphase
  - Kontrollflussorientierte Metriken wie Anweisungsüberdeckung, Zweigüberdeckung, Pfadüberdeckung oder Bedingungsüberdeckung

Zeilenmetriken

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

McCabe Cyclomatic Number

- Metrik zur Messung der Programmstruktur -> Ableitung der Komplexität eines Programmes -> Komplexitätszahl

- Bestimmte Schlüsselwörter (IF, SELECT Strukturen etc.) erhöhen die „Komplexitätszahl“
  COBOL&Co.: In Legacy Sprachen gibt es teilweise Kontrollstrukturen, die ggf. nicht mit dieser Metrik berücksichtigt werden, deshalb individuelle Berücksichtigungen notwendig/wünschenswert (PERFORM, EXEC, EVALUATE)

- Messung über Funktion, Subroutinen und Programm
  COBOL&Co.: Paragraph und Section

- Beispiel

Halstead Metriken (1)

- 1977 von Halstead vorgestellt, Umfangsmetrik
- Kennwerte:
  - Programmvolumen
  - Schwierigkeit, das Programm zu verstehen
  - Programmieraufwand (Schätzung)
  - Anzahl der zu erwartenden Programmfehler (Schätzung)
- Nachteil(e): Quellcode als zentrales Element wird überbetont, dadurch ggf. zu starke Vereinfachung

Halstead Metriken (2)

- Programmlänge (N)
  = Anzahl der Operatoren (N1) + Anzahl der Operanden (N2)

- Vokabulargröße (n) =
  = Anzahl versch. Operatoren(n1) + Anzahl versch. Operanden (n2)

- Programmvolumen/Halstead Volumen (V)
  = Programmlänge(N) * log2(Vokabulargröße(n))

- Schwierigkeitsgrad/Difficulty Level(D)
  = (Anzahl versch. Operatoren(n1) / 2) *
  (Anzahl der Operanden(N2) / Anzahl versch. Operanden(n2))

- Programmniveau/Program Level(L)
  = 1 / Schwierigkeitsgrad (D)

Halstead Metriken (3)

- Implementationsaufwand / Effort to Implement (E)
  = Halstead Volumen (V) * Schwierigkeitsgrad (D)

- Implementierzeit (T)
  = Implementieraufwand (E) / 18 (Ergebnis in Sekunden)
  (Schätzung)

- Anzahl der ausgelieferten Bugs (B) = Implementieraufwand (E) $^{2/3}$ / 3000
  (Schätzung)

- Zusammenfassung:
  - das nach Halstead ermittelte Maß korreliert stark mit der Häufigkeit von Programmierfehlern
  - Das Verfahren ist unabhängig von einer Programmiersprache einzusetzen

- Beispiel

Kennzahlenauswertung Clustering (1)

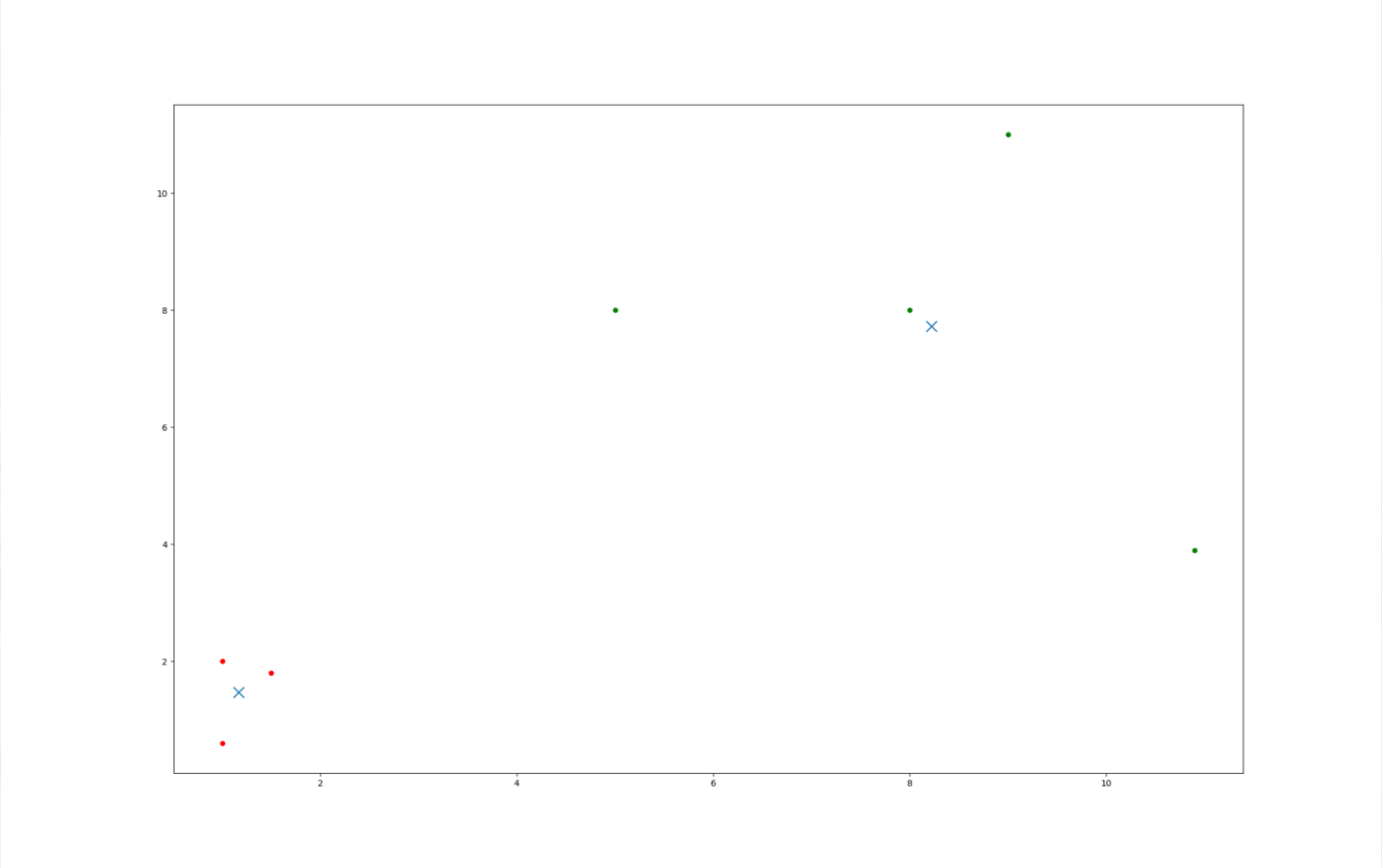
- Aus den Metriken werden Parameter für Komplexität und Aufwand abgeleitet
- Auswertung durch Clustering – Verfahren zur Entdeckung von Ähnlichkeitsstrukturen
  Im Beispiel = Suche nach ähnlich komplexem Quellcode
- Ziel: Einteilung und Priorisierung
- Lösungsansatz -> partitionierendes Clusterverfahren (K-Means)

Kennzahlenauswertung Clustering (2)

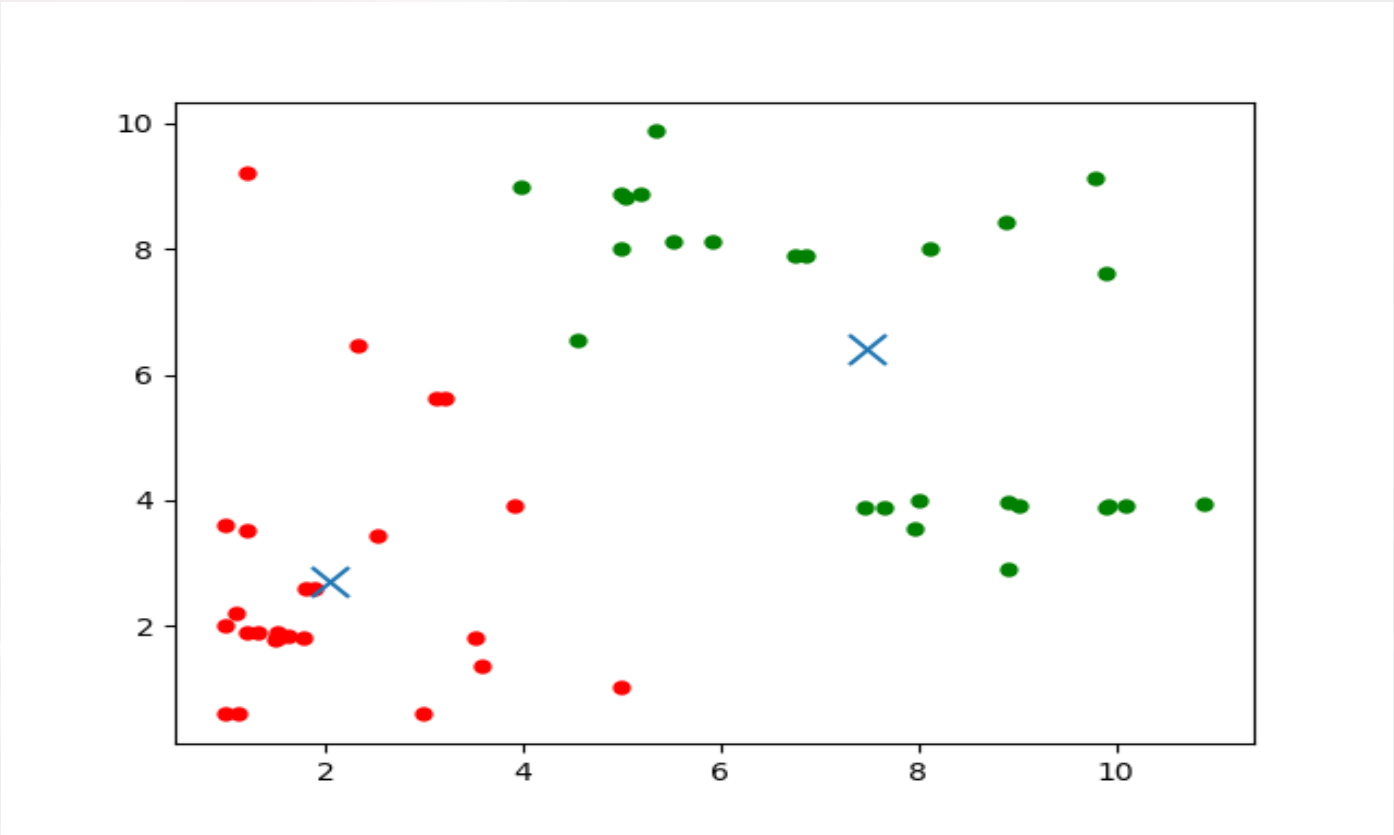
- Ziel von k-Means ist es, den Datensatz so in k Partitionen zu teilen, dass die Summe der quadrierten Abweichungen von den Cluster-Schwerpunkten minimal ist
- Entsprechende Funktion: $J = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$
- Ablauf (Lloyd – Algorithmus) – Umsetzung Scikit-Learn

1. Initialisierung, wähle k zufällige Mittelwerte aus den aus dem Datensatz
2. Zuordnung, jedes Datenobjekt wird demjenigen Cluster zugeordnet, bei dem die Cluster-Varianz am wenigsten erhöht wird.
   $S_i^{(t)} = \left\{ \mathbf{x}_j : \|\mathbf{x}_j - \mathbf{m}_i^{(t)}\|^2 \leq \|\mathbf{x}_j - \mathbf{m}_{i^*}^{(t)}\|^2 \text{ für alle } i^* = 1, \ldots, k \right\}$
3. Aktualisieren, berechne die Mittelpunkte der Cluster neu
   $\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$

Kennzahlenauswertung Clustering (3)



Kennzahlenauswertung Clustering (4)



Praktische Anwendung der Ergebnisse (1)

- Clustereinteilung zur Aufgabenpriorisierung (Welche Sourcecodes müssen geändert werden?)

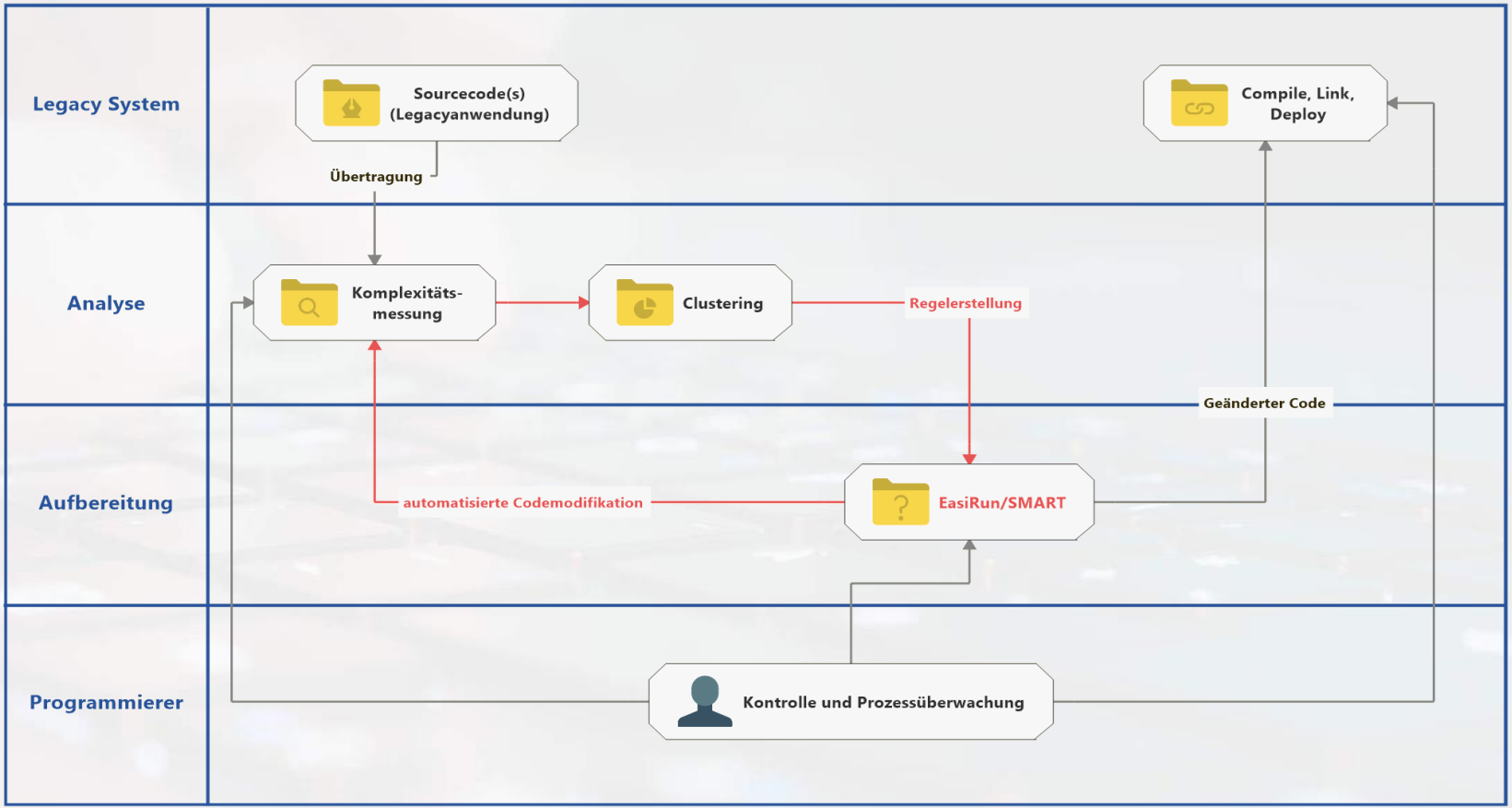
- Automatisierte Aufbereitung des Quellcodes zur Reduzierung der Komplexität (EasiRun Smart)

  - Regelbasiertes System – Regelerstellung auf Basis der Clusteringergebnisse
  - Sprachunabhängig
  - Steuerung über Trigger

- Ergänzung von Statischen Analyseprozessen zur Bestimmung des „Status Quo“ eines Legacysystems (KPI‘s)

- Beispiel

Praktische Anwendung der Ergebnisse (2)



Praktische Anwendung der Ergebnisse (3)

- Weitgehend automatisiertes Verfahren zur Softwarewartung auf Basis eines Clusteringprozesses
  - Reduzierung der Komplexität der Altanwendung
  - Schaffung von Ansatzpunkten für spätere Modernisierungsaufgaben (Microservices etc.)
  - Entlastung der Programmierer von „Alltagsaufgaben“ in Wartung bestehendem Quellcodes und bei Erweiterung
  - Hilfestellung beim Wissenstransfer für Neueinsteiger durch replizierbare Ergebnisse

Praktische Anwendung – weitergedacht!

Aufbereitung der Clustering – Ergebnisse automatisiert für Ticketingsysteme
-> „ToDo“ Liste für den/die Programmierer

Dynamische Statusberichte generieren

Schaffung eines „clusterbasierenden“ Abrechnungsmodells und eines damit einhergehenden Abrechnungssystems

Noch mehr Ideen? -> ugraf@easirun.de

Fragen(?) und/oder Anregungen(!)

ugraf@easirun.de