ASERVO Software

KNOW-HOW

SLAs und KPIs für CI/CD

Gute Metriken im DevOps-Umfeld finden und definieren

Gute Metriken im DevOps-Umfeld finden und definieren

Continuous Integration (CI) und Continuous Delivery (CD) sind grundlegende Bestandteile einer agilen Softwareentwicklung. Durch sie soll schnell gute Software entstehen. Automatisierte, zugleich flexible Prozesse sind dabei für Entwicklung und Betrieb (DevOps) essentiell. Wie aber lassen sich die Erfolgsfaktoren agiler Projekte definieren? Messen SLAs und / oder KPIs den Erfolg? Welche Messgrößen machen wann Sinn?

Anwendbarkeit von KPIs und SLAs auf CI/CD //

Betreibt ein Unternehmen CI/CD als zentralen Service, sind für diesen Service - wie für andere Services des Unternehmens - üblicherweise SLAs vorgegeben. Diese umfassen meist Vorgaben für **Verfügbarkeit, Performance** oder auch **Reaktionszeiten** auf Incidents. SLAs stammen ursprünglich aus dem IT-Umfeld und finden insbesondere im Bereich Hosting seit längerem Einsatz. Dort sind die Rahmenbedingungen für die Bereitstellung eines Service hinreichend gut verstanden und definierbar.

Für KPIs gilt dies nicht gleichermaßen. Sie für einen Anwendungsservice zu formulieren ist generell nicht ganz so einfach. In einem agilen bzw. DevOps-Umfeld müssen sich darüber hinaus auch KPIs einem ständigen Feedback unterwerfen. Diese **kontinuierliche Rückkoppelung** modifiziert die zunächst gewählten KPIs fortlaufend. Über die Zeit und mit steigender Reife des CI/CD-Service treten etablierte Metriken in den Hintergrund und andere - meist "höherwertige", komplexere - gewinnen an Bedeutung. Das ist normal und kein Grund zur Panik.

KPIs dienen dann häufig dazu, Bewertungen und/oder Boni von Mitarbeitern zu steuern. Über lange Zeit kamen diese Vorgaben aus den oberen Hierarchieebenen. Immer öfter erarbeiten aber auch DevOps Teams gemeinsam KPIs, die dann einem ständigen Feedback unterliegen, so dass sie sich im laufenden Betrieb auch ändern können.

Begriffsdefinitionen

CI (Continuous Integration) bezeichnet das Übersetzen und Testen von Software nach jedem Commit / Push. Das Endresultat ist in der Regel ein binäres Artefakt, das in einem Repository zur weiteren Verwendung abgelegt wird.

CD (Continuous Delivery / Deployment), als Superset von CI, testet das Zusammenspiel der erzeugten Artefakte mit dem Ziel, Produktionsreife zu erlangen. Continuous Delivery stellt die entsprechenden binären Artefakte für das Deployment bereit, bei Continuous Deployment werden sie - entsprechend erfolgreiche Tests vorausgesetzt - automatisch produktiv gesetzt.

SLAs (Service Level Agreements) gibt es in der einen oder anderen Form bereits sehr lange aus dem Bereich Hosting. Sie bezeichnen zugesicherte Merkmale einer Leistung, auf die sich Auftraggeber und Auftragnehmer vor Erbringung der Leistung geeinigt haben. Sie haben Ähnlichkeiten mit Vertragsbedingungen oder zugesicherten technischen Eigenschaften. .....

Das oberste Mantra für DevOps ist 'Messen', d. h. es wird eine Vielzahl von **Metriken** erfasst und mit Hilfe von geeigneten Werkzeugen ausgewertet.

Für CI/CD als Service bedeutet das, eine Vielzahl an Metriken von einer Vielzahl an Systemen zu erfassen:

- Vorsystemen wie z. B. SCM, LDAP, Mail, HTTP Proxy, Ticketing
- Infrastrukturen wie Build Server, Agents, Test-Maschinen
- Performance-Messwerte der Anwendung in Produktion

Sind alle diese Messwerte erfasst, geht es an die Arbeit...

....KPIs (Key Performance Indicators) sind - allgemein gesprochen - Messwerte für die Erreichung von Zielen. Sie sollen Auskunft darüber geben, wie gut oder schlecht die gemessenen Werte im Vergleich zu vorgegebenen Zielen oder zu Durchschnittswerten vergleichbarer Unternehmen liegen.

DevOps (Development + Operations) bezeichnet eine Vorgehensweise aus dem agilen Umfeld, die Entwickler, Tester und Infrastruktur-Betreiber in einem Team zusammenfasst ("You build it, your run it, you fix it").

Messgrößen richtig definieren

//

SLA-Definitionen müssen auf die gemessenen Daten abgebildet werden: Bedeutet "verfügbar", ob das betreffende System überhaupt verfügbar ist, oder dass es z.B. innerhalb einer definierten maximalen Zeit auf einen definierten Request antwortet? Das entspricht der Erarbeitung einer "Definition of Done" (DoD) aus dem agilen Umfeld.

Auch für KPIs gilt es eine Entsprechung in den erhobenen Daten zu finden. Ein **"Indicator"** ist kein absoluter Messwert. Ein Indicator ist eine Aufforderung, genauer hinzuschauen. Gibt es Abweichungen (meist auf der Zeitachse), muss man sich immer den Grund anschauen und nicht einfach den Wert akzeptieren.

DoD DoD ist die englische Kurzform für eine Liste von Kriterien, die ein Produkt erfüllen muss, um als fertig zu gelten.

Warum KPIs nicht ganz so einfach sind

//

In größeren Unternehmen gibt es die Tendenz, aus KPIs **direkt** Beurteilungen oder variable Gehaltsbestandteile (Boni) abzuleiten. Das ist häufig zu kurz gesprungen. Viele der Werte aus der Übersicht unten klingen je nach Blickwinkel zunächst plausibel, offenbaren aber bei näherem Hinsehen und unter Einbeziehung der menschlichen Natur einige Schwächen.

Beispiele:

Lines of Code pro Entwickler pro Tag - kam so tatsächlich von einer hochbezahlten Beratungsfirma und wurde zum Glück wegen offensichtlichen Unfugs verworfen.

Kostenverteilung nach Nutzung - Möchte ich einen Service etablieren, sollte ich nicht für die Nutzung zahlen lassen sondern eher die Nichtnutzung strafen, also etwa die Kosten für den Service allen in Rechnung stellen. Wer den Service nicht nutzt, gerät in Rechtfertigungsprobleme.

Dauer eines Builds - Die Dauer eines Builds wird von zu vielen unterschiedlichen Faktoren beeinflusst, wie z. B. Anzahl und Gründlichkeit von Tests, Parallelisierung innerhalb des Builds, Verfügbarkeit von Ressourcen usw.

Anzahl der Fehler einer Komponente in einer Iteration - Kein guter Indikator, weil er zu stark von den Individuen und den Umgebungsbedingung abhängig ist. Eventuell aber gut, um den Prozess zu verbessern wie z. B. Commits / Pushes nur dann, wenn lokal alle Tests durchlaufen wurden.

Anzahl Tests - Die Anzahl der Tests lässt sich leicht steigern, ohne dass dadurch die Qualität steigt.

Testabdeckung - Taugt als alleiniges Kriterium nur sehr bedingt. Wichtiger ist, dass der Wert kontinuierlich besser wird. Wichtig ist aber auch eine gemeinsame Definition dessen, was wie getestet werden soll.

Durchlaufzeit eines Tickets - Führt in der Regel dazu, dass Tickets gnadenlos geschlossen werden, ohne dass das eigentliche Problem behoben ist. Besser ist eine Kombination aus Messwerten, die sowohl die Schritte innerhalb des Wokflow inklusive Schleifen als auch andere Faktoren berücksichtigen.

Gefundene Fehler in Produktion - Hier ist eine Analyse besser, warum Fehler erst in Produktion gefunden werden

Abgeschaltete Tests / Anzahl der Tests pro Release - Bei Auffälligkeiten ist es ein guter Anlass, sich die Ursachen anzuschauen: Wird der Code gerade refactored, werden neue Third-Party-Libraries verwendet, so dass einige der existierenden Tests nicht ohne weiteres anzupassen sind? Auch hier ist aber eher der Vergleich zum vorigen Release interessant.

Architectural Index / Maintainability Index (z. B. aus SonarQube) - Ein sehr guter Indikator für die Code-Qualität, aber nicht für andere Aspekte der Anwendung.

Anzahl bekannter Sicherheitslücken pro Release pro Anwendung, aufgeschlüsselt / gewichtet nach Severity. Realistisch sollte man hier nur die Verbesserung messen und nicht den absoluten Wert.

Auslastung der Infrastruktur - Je nach zur Verfügung stehenden Ressourcen macht es generell Sinn, die Auslastung zu messen. Allerdings hängt die Interpretation von vielen Details ab, z. B. muss ich eine statische Infrastruktur mit Bare Metal oder VMs in dieser Hinsicht anders bewerten als einen Kubernetes-Cluster.

Visualisierung von KPIs - ausgewählte Beispiele
//

Die folgenden Abbildungen zeigen Beispiele, die mit einer Kombination aus **Prometheus** und **Grafana** entstanden sind. Üblich in diesem Kontext ist auch die Nutzung des ELK-Stacks (Elasticsearch, Logstash, Kibana).

Prometheus ist ein Open Source Toolkit für System-Monitoring, Alerting und -Trending auf der Basis von Zeitreihen. https://prometheus.io/

Grafana ist ein Open Source Monitoring Programm, mit dem sich Daten aus verschiedenen Datenquellen analysieren, aufbereiten und grafisch darstellen lassen. https://grafana.com/

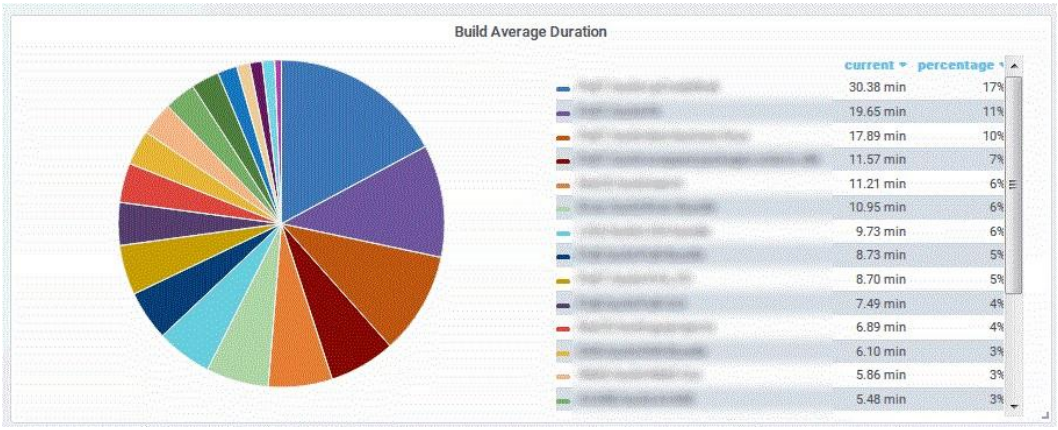

Abb. 1: Durchschnittliche Build-Dauer

Die Grafik aus Abbildung 1 ist gut geeignet für einen ersten Überblick, es gilt aber verschiedene Dinge zu hinterfragen bevor daraus KPIs ableitbar sind:

- Sind die Builds homogen, d. h. machen alle Builds strukturell dasselbe oder gibt es eine bunte Mischung aus Microservices, J2EE und C#?
- Wie ist das Delta zu früheren Zeitpunkten? Was ist der Erwartungswert von Seiten der Entwickler?

Abb. 2: Build Resultate

Auch die Grafik in Abbildung 2 eignet sich gut für einen ersten Überblick, die Ergebnisse sind aber ohne Kenntnis der Zusammenhänge nicht aussagekräftig:

- Wird testgetrieben gearbeitet? Je nach Vorgabe ändert sich die Erwartungshaltung, welcher Anteil der Builds erfolgreich sein soll.
- Was sind die Ursachen für fehlgeschlagene Builds? Infrastruktur- oder Programm-Probleme?

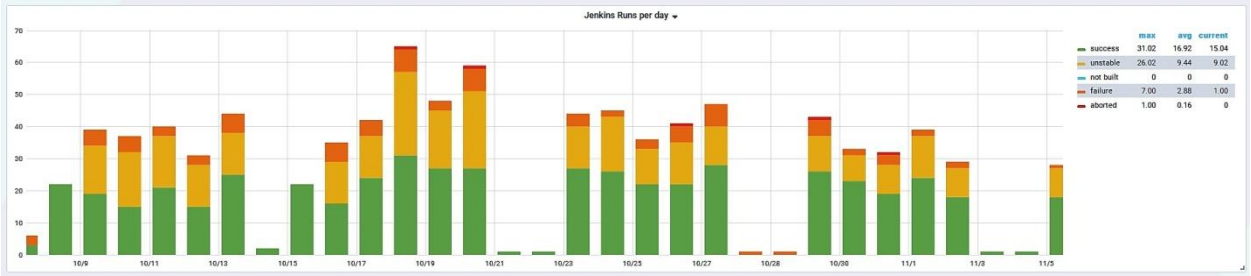

Abb. 3: Builds pro Tag

Die Builds pro Tag wie in Abbildung 3 zu sehen, bieten einen guten Einstieg für den täglichen Kontrollblick des Service-Betreibers.

- Eine plötzliche Häufung von fehlgeschlagenen Builds gibt Anlass für weitere
- Nachforschungen.
- Bleibt das Verhältnis zwischen erfolgreichen, instabilen und gescheiterten Builds einigermaßen konstant, läuft der Service im Wesentlichen rund.

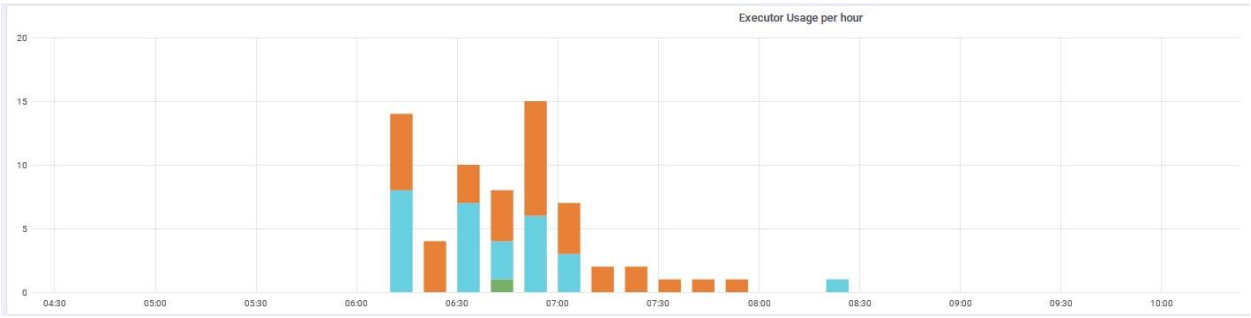

Abb. 4: Executor Usage pro Stunde

Die Executor Usage pro Stunde aus Abbildung 4 bietet eine wichtige Auswertung, wobei aber auch hier der Kontext zu beachten ist:

- Habe ich von einer bestimmten Sorte Exekutoren eventuell nur eine beschränkte Anzahl? Das sollte ich separat messen.
- Habe ich ein Limit auf der maximalen Anzahl von Exekutoren z. B. auf Grund der Infrastruktur ? Auch das sollte ich separat messen.
- Normalerweise ergibt sich bei CI eine typische Verteilung zwischen zeitgesteuerten Builds und Push- / Commit-gesteuerten Builds. Hierbei sollte man darauf achten, dass es möglichst keine Überschneidungen gibt. Die Builds aus dem Tagesgeschäft häufen sich meist vor der Mittagspause und vor dem Feierabend, also sollten zeitgesteuerte Builds in den frühen Morgen- oder späten Abendstunden stattfinden.

Abb. 5: Queued Builds

Queued Builds wie in Abbildung 5 zu sehen, sind ein Zeichen dafür, dass nicht genug Exekutoren zur Verfügung stehen.

- In diesem Fall ist es ein Nightly Build, der viele Komponenten baut. Nachts dürfte das niemanden stören, aber am Tage wären wertvolle Ressourcen blockiert.
- Queue Peaks können auch dadurch entstehen, dass alle Master gleichzeitig auf den Agent Pool zugreifen wollen.
- Eine weitere Ursache kann sein, dass von einer bestimmten Sorte Agents nicht genügend viele vorhanden sind.

Worin besteht denn aber das eigentliche Ziel von CI/CD? Hält man sich diese Frage vor Augen, so lautet die Antwort in der Regel, **Software in guter Qualität** und hoher Geschwindigkeit zu produzieren. Zu guter Qualität gehören z. B. Wartbarkeit, Performance, Ausschluss von bekannten, kritischen Sicherheitslücken, Einhaltung von Standards aus dem Umfeld **Governance, Risk und Compliance**.

Für jeden einzelnen dieser Begriffe müssen sich alle Beteiligten - ob Betreiber, Nutzer, Service-Sponsoren oder andere - **im Vorfeld** auf eine gemeinsame Sicht verständigen und im Zweifel im Laufe des Projekts diese Sicht anpassen.
Um Software unter dieser Prämisse entwickeln zu können, braucht es das Zusammenspiel mehrerer Tools:

- SCM (z. B. Git, Subversion, Mercurial)
- Ticketing (z.B. Jira)
- Build (z. B. Jenkins)
- Code-Analyse und Test-Auswertung (z. B. SonarQube)
- Schwachstellenanalyse (z.B. Nexus Lifecycle, Nessus)
- Unit Tests, Integration Tests, User Acceptance Tests, Performance Tests, Regressions-Tests
- Application Performance Monitoring

Als Teile eines zentralen CI/CD Service liefern alle diese Systeme Messwerte, die für KPIs und die Kontrolle von SLAs genutzt werden können.

Welche dieser Messwerte tatsächlich relevant sind, hängt von vielen spezifischen Details ab. Meist ist es sinnvoll, mit einer Handvoll einfacher Werte zu starten und diese dann weiter zu verfeinern, wenn man die ersten Auswertungen mit echten Daten gesehen hat. Wichtig ist auch festzulegen, **was** man messen möchte.

Wie finde ich die passenden KPIs?
//

Es gibt nicht **den einen Satz** von KPIs, der auf jedes beliebige Setup passt. Vielmehr ist es so, dass je nach Kunde, verwendeten Tools und verwendeter Technologie spezifische KPIs ermittelt werden müssen. Am besten ist es, mit einigen einfachen KPIs zu beginnen und sie mit steigender Erfahrung so zu modifizieren, dass sie zum **Einsatzzweck** passen.

Business KPIs
//

Aus Sicht des Business gibt es zwei zentrale KPIs:

1. "Idea to Production" oder auch **"Time to Market"** - die Zeit zwischen der Formulierung einer Idee als Ticket und dem Going-Live des Features. Hier fließen mehrere Faktoren ein:

- Wie präzise die Idee im Ticket festgehalten wurde (Beschreibung, Akzeptanzkriterien)
- "Dicke" des Tickets (kleine Änderung / Ergänzung vs. Wechsel der Architektur)
- Priorisierung / Auslastung der Entwickler
- Geschwindigkeit der CD-Strecke
-

2. "Hotfix deployment" oder auch **"MTTR (Mean Time To Repair)/ MTTF (Mean Time To Fix)"** - die Zeit zwischen der (Analyse eines Problems und der) Erstellung eines Hotfixes und dem Going Live. Auch hier gibt es mehrere Faktoren:

- Qualität und Umfang der vorhergegangenen Analyse
- Geschwindigkeit der CD-Strecke vs. Vollständigkeit der Tests

Erfahrungsgemäß ist es sehr sinnvoll, sich bereits frühzeitig Gedanken um das Hotfix Deployment zu machen (Auf welche Tests verzichte ich? Spezielle Pipeline oder spezielle Parameter der "normalen" Pipeline?), damit man im Ernstfall nicht in Panik verfällt und Fehler macht.

Weitere Messwerte, die für den Betrieb eines zentralen Service relevant sein können:

Change Success Rate
Die Anzahl erfolgreicher Builds / Deployments im Verhältnis zur Gesamtzahl aller Builds / Deployments. Ist die Change Success Rate zu gering, sollte man analysieren, wo in der Pipeline der Fehler liegt. Können Sourcen nicht kompiliert werden, liegt die Schuld in der Regel beim Committer. Scheitert die Pipeline beim integrativen Testen, braucht man eventuell bessere Mocks, um solche Fehler bereits früher abzufangen. Scheitert die Pipeline an Quality Gates, sind deren Daten evtl. nicht in der IDE des Entwicklers verfügbar oder er weiß nicht, was er mit der vorhandenen Information anfangen soll.

Deployments pro Monat pro Pipeline / Anwendung
Ermöglicht die Vergleichbarkeit unterschiedlicher Anwendungen und Technologien, sofern die Rahmenbedingungen einigermaßen gleich ausfallen.

"Lead time for change"
Wie lange braucht ein Commit bis er PROD erreicht (Minimum, Maximum, Average)? Ist verwandt mit 1. und 2.

"Batch size"
Wieviele Story Points pro Deployment (Minimum, Maximum, Average)? Das hängt fallweise mit der Velocity aus Scrum zusammen.

Application KPIs
//

Code-Qualität
Auswertung von Testabdeckung, Maintainability Index, Architectural Index usw., in der Regel als Delta über die Zeit oder gegen gesetzte Standards. Abgeleitete Indices wie z. B. Maintainability oder Architectural Index sind weniger anfällig gegenüber Manipulationen als die einfachen Metriken wie etwa Testabdeckung. Auf jeden Fall muss zuerst das Messverfahren abgestimmt werden - sollen z. B. Getter/Setter getestet werden, wie sieht es aus mit generiertem Code?

Kritische Security-Bugs
Gesamtzahl bzw. Anzahl in neuem / geändertem Code eines bestimmten Levels. Hier kann auch die Unternehmenssicherheit bestimmte einzelne Fehler besonders herausheben.

Performance-Abweichungen
Sind immer mit Vorsicht zu genießen, sollten aber auf jeden Fall beobachtet werden. Gibt es unerwartete Abweichungen zu früheren Messwerten, sollte auf jeden Fall die Ursache ermittelt werden.

System KPIs
//

Verfügbarkeit
Zu wie viel Prozent steht der Service zu den vorher vereinbarten Zeiten zur Verfügung (7 x 24 vs. 5 x 9)? Gibt es vorher definierte Wartungszeiträume auf Seiten der Infrastruktur oder des Service?

Fehlerhafte vs. erfolgreiche Aufrufe
Wann liefert der Service Fehler? Das kann z.B. bei Session Timeout oder Deep Links passieren. Bei manchen Anwendungen funktionieren Deep Links nicht gut, dann gilt es andere Wege zu finden, um die gewünschte Funktionalität bereitzustellen.

Queue Wait Time (Minimum, Maximum, Average)
Wie lange muss ein Job durchschnittlich / maximal warten, bis er "dran" ist? Wenn Wartezeiten entstehen, was ist die Ursache? Gibt es generell zu wenige Agents, gibt es zu wenige Agents einer bestimmten Sorte, starten alle Nightly Builds zum selben Zeitpunkt?

Builds/Deployments pro Tag/Woche
Tatsächliche, sinnvolle Werte sind abhängig von der Applikation und der Art des Deployments. Auch hier ist - je nach Ziel - hauptsächlich das Delta über die Zeit interessant; in der Regel ist das Ziel, mehr Deployments pro Zeiteinheit zu schaffen.

Auslastungsgrad der Build Agents
Da unterscheiden sich Setups mit statischen Maschinen grundsätzlich von dynamischen Infrastrukturen wie z. B. Docker / OpenShift / Kubernetes / AWS / Azure / usw. Für statische Maschinen strebe ich eine möglichst gleichverteilte Auslastung an. Beim Arbeiten mit dynamisch bereitgestellter Infrastruktur geht es eher um Limitierung bzw. Kappung der Kosten.
.

Prozess KPIs
//

Prozessbezogene KPIs sind Indikatoren dafür, wie gut Prozesse gelebt werden:
WTFs pro Tag/Woche
Wie oft erlebt das Team WTF ("What the fuck") - Momente? Wie oft tauchen Dinge auf, mit denen vorher niemand gerechnet hatte?

Impediments pro Sprint
Wie viele Impediments kommen pro Sprint ans Licht?

Impediment Removement Time
Wie lange dauert es, bis ein Impediment beseitigt wird?

Nichtverfügbarkeit des Product Owners
Ein häufiges Problem, wenn Agiles Arbeiten eingeführt wird: Projektmanager werden zu Product Ownern umgewidmet, aber ansonsten ändert sich nur wenig. Das führt automatisch dazu, dass sie der Aufgabe eines Product Owners nicht gerecht werden können - weder aus Sicht des Unternehmens noch aus Sicht des / der Teams.

Zusammenfassung
//

SLAs sollte man vor jeder Inbetriebnahme eines Services zwischen Betreiber und Nutzer vereinbaren, damit es später keine **Missverständnisse** wegen völlig unterschiedlicher Erwartungen gibt.

KPIs sind Indikatoren, bei deren Veränderung in der Regel genaueres Hinschauen erforderlich ist. Sie eignen sich nur schlecht 1:1 zur Messung von Qualität, meist ist die

Differenz zu einem vorigen Zeitpunkt der bessere Ansatz. Sie sollten regelmäßig reevaluiert und überarbeitet werden.

Es gibt unterschiedliche Arten von KPIs, je nach **Blickwinkel**; jeder dieser Blickwinkel hat seine Berechtigung. Häufig besteht die Gefahr, sich zu sehr auf die rein technischen Messgrößen einzulassen. Erfahrungsgemäß bringen aber gerade die Anwendungs- und Prozess-KPIs am meisten Erkenntnisgewinn, obwohl ihre Ermittlung mit mehr Aufwand verbunden ist. Die technischen KPIs helfen eher bei der Diagnose und Beseitigung von Schwächen.

ASERVO Software GmbH
Konrad-Zuse-Platz 8
81829 München

Tel.: +49 (0) 89 7 16 71 82-40
Fax: +49 (0) 89 7 16 71 82-55
Email: rmayr@aservo.com

www.aservo.com