# 3. Systemlandschaft

> The problem when solved will be simple.
>
> *(Charles Kettering, Erfinder)*

Inhaltsangabe



Dieses Kapitel besteht aus vier Abschnitten. Zuerst wird ein Überblick auf das MeDIC-Projekt gegeben, welches die Grundlage für diese Arbeit darstellt. Anschließend wird der Aufbau des MeDIC-Dashboards erklärt und seine besonderen Merkmale herausgestellt. Daran anknüpfend soll das Konzept der geplanten Mess-Infrastruktur mit seinen verschiedenen Schichten erläutert. Schließlich wird die Notwendigkeit einer Erweiterung des Modells um Variabilität erörtert.

## 3.1. MeDIC – Metric-Based Project and Process Management

*MeDIC* ist eine Werkzeugsammlung zur Unterstützung des Projektmanagements von Softwareprojekten und wird zur Zeit von der Forschungsgruppe Softwarekonstruktion entwickelt. Den Hauptprogrammteil bildet das sogenannte *MeDIC-Dashboard*, welches in systematischer und geordneter Weise projektspezifische Messdaten bzw. Metriken visualisiert. Die erste Version von MeDIC entstand im Jahr 2012 im Rahmen der Masterarbeit von Frederic Evers [13] und wurde fortan, auch im Rahmen von Folgearbeiten, stetig weiterentwickelt (siehe [23, 51, 27, 11, 33]).

Das Akronym MeDIC steht für **Me**tric **D**ocumentation, **I**ntegration and **C**onfiguration und deutet damit die zentralen Teilaspekte des Systems an.

- **Metric Documentation**
  Es unterstützt die Organisation und Dokumentation der im Unternehmen eingesetzten Metriken, indem es eine zentrale Verwaltungseinheit zu Verfügung stellt, die eine systematische Katalogisierung und Kategorisierung von

Metrik-Spezifikationen und deren Beschreibungen bzw. Interpretationshilfen bereitstellt (siehe Abschnitt 5 Metrikmanagement-Datenbank).

- **Metric Integration**
  Die Integration heterogener Systeme in MeDIC und die damit verbundene Datenanbindung ermöglicht eine Verknüpfung zwischen Messgegenstand und Metrik (siehe Abschnitt Messarchitektur 3.3).

- **Metric Configuration**
  MeDIC hat die Zielsetzung Metriken, deren Repräsentation sowie die benutzerdefinierte Zusammenstellung von Dashboards frei konfigurierbar zu gestalten. Gleichzeitig soll dem unerfahrenen Dashboard-Benutzer durch sinnvolle Vorlagen und Voreinstellungen der Einstieg erleichtert werden (siehe [27]). Das Konzept der Wiederverwendung spielt eine wichtige Rolle bei der Dashboard-Einrichtung.

## 3.2. MeDIC-Dashboard

Das MeDIC-Dashboard richtet sich primär an Projektleiter, die einen Überblick über die Abläufe und die Verfassung laufender Softwareprojekte erhalten möchten. Insbesondere Erkenntnisse über Probleme oder negative Trends innerhalb des Projekts sind von Belang.

Abbildung 3.1 zeigt die Dashboard-Ansicht der aktuellen Version für ein Beispielprojekt. Die Ansicht gliedert sich in die drei Unterbereiche Dashboard-Gitternetz (oben links), Informationsbedürfnisse (oben rechts) und Funktionsleiste (unterer Rand). Im Gitternetz befinden sich sogenannte *Dashboard-Container*, welche wiederum die *Dashboard-Items* enthalten, die jeweils ein Diagramm oder eine Tabelle anzeigen. Die Dashboard-Items sind die Grundbausteine eines Dashboards und sind für die Visualisierung der Projektmetriken verantwortlich. Der Dashboard-Container mit dem Titel „Wichtige Projektmetriken“ enthält beispielsweise sieben Dashboard-Items vom Typ `BulletGraph`, wohingegen der Dashboard-Container „Commits“ nur ein Dashboard-Item vom Typ `ChartesianChart` beinhaltet.

Wenngleich das Konzept des Dashboards nicht neu ist, führt MeDIC auf zweckdienliche Weise eine Reihe ausgereifter Konzepte zusammen, um die Benutzbarkeit für den einzelnen Nutzer zu erhöhen und ermöglicht somit insgesamt eine effektivere Arbeitsweise.

Folgende Besonderheiten werden in MedIC-Dashboard vereint:

**Visuelle Gestaltung**
Die Gestaltung orientiert sich zu großen Teilen an den Vorschlägen von Stephen Few, der in seinem Buch *Information Dashboard Design* [15] 13 häufige Fehler beim Dashboard-Design auflistet.

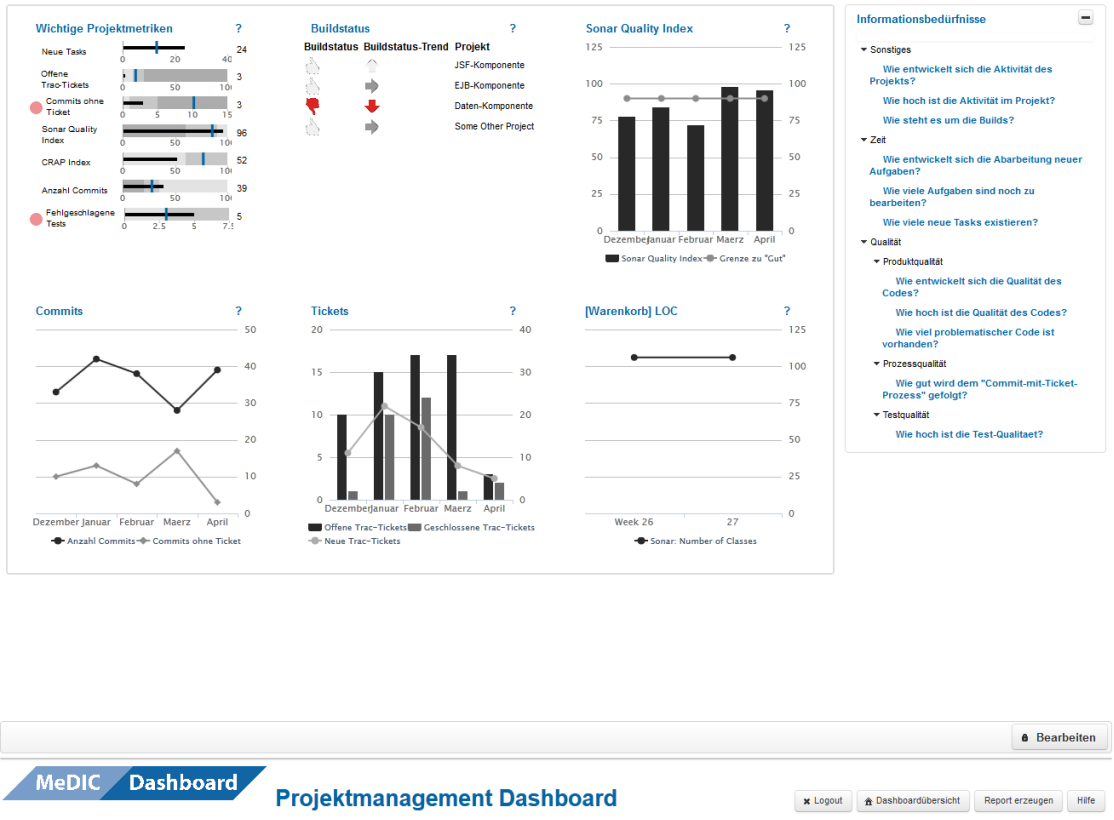


Abbildung 3.1.: MeDIC-Dashboard Bildschirmfoto

Die Farbgebung des MeDIC-Dashboards ist insgesamt sehr schlicht gehalten, so dass vorwiegend Grautöne Verwendung finden. Der Einsatz von Signalfarben wie rot ist zur Hervorhebung von wichtigen Hinweisen reserviert. Des Weiteren spielt die richtige Anordnung der Anzeigeinstrumente eine wichtige Rolle. Diese und viele andere Aspekte (nachzulesen in [13] und [15]) tragen dazu bei, den Fokus auf die wesentlichen Dinge zu lenken und die Benutzbarkeit zu erhöhen.

**Enge Kopplung mit Informationsbedürfnissen**

Die Dashboard-Items sind direkt mit sogenannten Informationsbedürfnissen[1] verknüpft. Die Grundidee besteht darin, dass Metriken (und folglich Dashboard-Items) bestimmte Informationsbedürfnisse erfüllen. Ausgehend von einer Menge an Informationsbedürfnissen lässt sich ein Dashboard individuell nach den jeweiligen Anforderungen zusammenstellen. Darüber hinaus lassen sich bereits angelegte Dashboards besser verstehen, da pro Dashboard-Item die Informationsbedürfnisse in Form von Fragen über eine Schaltfläche abgerufen werden können. So kann genau nachvollzogen werden, welchem Zweck die

[1] Definition nach [28]: Insight necessary to manage objectives, goals, risks and problems.

angezeigten Visualisierungen dienen.

**Wissensmanagement**
Eine integrierte Hilfefunktion ermöglicht den Zugriff auf ein System zum Wissensmanagement. Hier können Erfahrungen und Interpretationshilfen für Dashboard-Items ausgetauscht werden. Aktuell beschäftigt sich Driss El Majdoubi in seiner Diplomarbeit [10] außerdem mit der Entwicklung einer Erklärungskomponente für MeDIC.

**Dashboard-Initialisierung**
Die regelbasierte Initialisierung von Projektdashboards [11] sowie der Entwurf und die Evaluierung von Dashboard-Vorlagen [27] wurden konzeptionell behandelt und bedürfen nach heutigem Stand noch einer endgültigen technischen Integration.

**Generierung von Berichten**
Eine Exportfunktion von Diagrammen und die Generierung von Berichten sind aktuell Forschungsgegenstand der Diplomarbeit von Paul Tokarev [54].

## 3.3. Messarchitektur

Das Konzept für eine Messarchitektur zur Integration heterogener Systeme in MeDIC wurde erst kürzlich von Andreas Steffens in seiner Diplomarbeit [51] entworfen und wird aktuell im Rahmen weiterer Überarbeitungen in MeDIC integriert.

Die Problemstellung mit der sich viele Projektleiter großer Organisationen konfrontiert sehen ist die Tatsache, dass sie Informationen aus vielen verschiedene Quellen benötigen. Um dem Anspruch der Kontrolle von Kosten, Qualität, Zeitplan, Budget etc. gerecht zu werden, müssen viele verschiedene Informationsbedürfnisse befriedigt werden. Jedoch werden die jeweiligen Informationen gewöhnlicherweise in verschiedenen Systemen gespeichert [59].

In Abbildung 3.2 ist das Messarchitekturmodel *Enterprise Measurement Infrastructure* (EMI) zu sehen, welches für MeDIC entworfen wurde. Es erlaubt die Integration heterogener Systeme und verfolgt einen service-orientierten Ansatz mit einer klaren Trennung von Systemintegration, Berechnung und Visualisierung.

Das Schichtenmodell zeigt an unterster Stelle die *Data-Provider*-Schicht. Hier handelt es sich um die zu integrierenden Systeme, die wertvolle Informationen besitzen, die für einen Projektleiter von Interesse sein könnten.

Spezielle Datenadapter der *Data-Adapter*-Schicht hören auf bestimmte Ereignisse[2] und schicken für die Berechnung von Metriken relevante Daten auf den *Enterprise Measurement Data Bus* (EMDB) der Data-Transport-Schicht. Normalerweise wird ein solches Ereignis ausgelöst, wenn ein Datenlieferant aktualisierte Daten zur Verfügung

[2]Die Ereignisse werden nach dem Invoke-Push-Mechanismus ausgelöst (siehe [59])

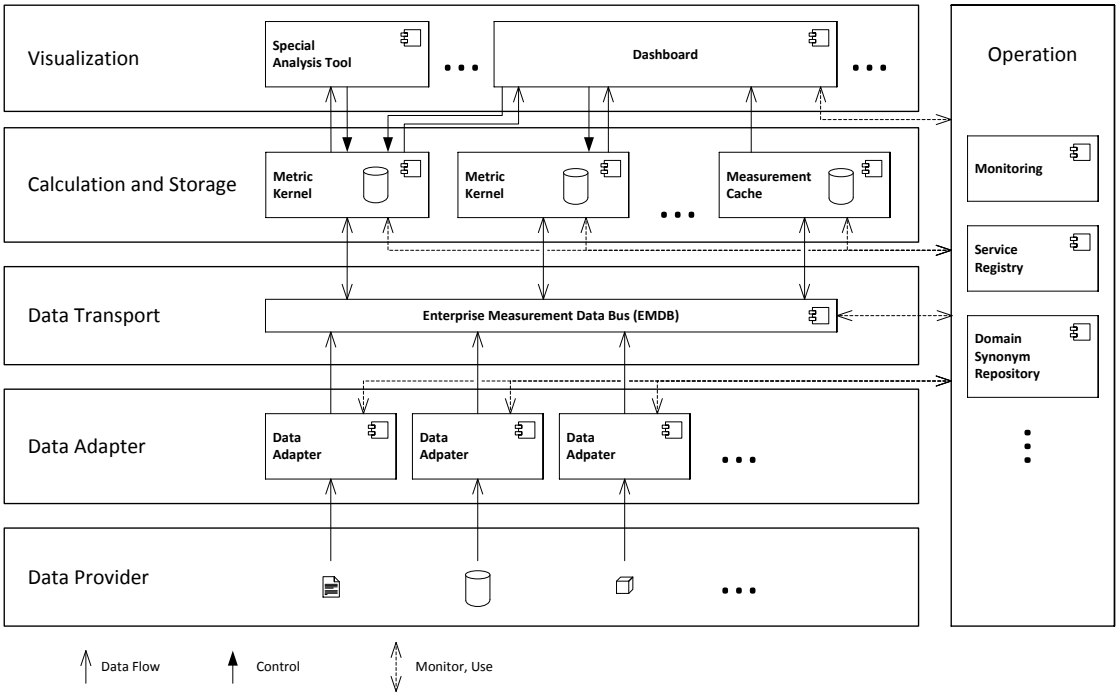


Abbildung 3.2.: Messarchitektur in MeDIC (aus [59])

hat. Beispielsweise könnte ein Bug-Tracking-System jedes Mal, wenn ein Ticket erzeugt oder geschlossen wird ein Ereignis auslösen, dass den Datenadapter dazu veranlasst, eine Nachricht mit eben diesem Ticket auf den EMDB zu schicken.

Der EMDB ist eine Implementierung des Enterprise Service Bus [47] und basiert auf der Java Message Service (JMS) API. Der Grundaufbau einer EMDB-Nachricht besteht aus folgenden drei Eigenschaften

- **metricRefId**: Dient der Einordnung der Nachricht zu einem bestimmten Mess- bzw. Nachrichtentyp.
- **eomId**: Über diesen Wert, kann die Nachricht einem Projekt (Entity of Measurement) zugeordnet werden.
- **value**: Der eigentliche Messwert.

Die *Rechenkerne*[3] (engl. Metric Kernel) der *Calculation-and-Storage*-Schicht können nach dem Publish-Subscribe-Verfahren bestimmte Nachrichtentypen abonnieren, die für sie zur Weiterverarbeitung relevant sind. Jeder Rechenkern, repräsentiert durch eine Sammlung von Java-Klassen, steht für eine bestimmte Metrik, die Messungen auf Basis der über den Bus gelieferten Daten erzeugt. Weiterhin besitzt jeder Rechenkern ein eigenes Datenmodell, um den Inhalt der EMDB-Nachrichten zu persistieren. Die

[3] auch Metrikmodell genannt

berechneten Messungen können ihrerseits ebenso auf den Bus gegeben werden und als Eingabe für andere Rechenkerne dienen. Der *Measurement Cache* soll dazu dienen die vorberechneten Messwerte zwischenzuspeichern. Das ist insbesondere dann von Vorteil, wenn bestimmte Rechenkerne auf die Berechnungen von anderen Rechenkernen angewiesen sind (und nicht nur auf Messungen, die direkt von den Datenadaptern geliefert werden). Ansonsten könnte möglicherweise eine Kaskade von zeitintensiven Berechnungen ausgelöst werden.

Schließlich kann das MeDIC-Dashboard, welches sich kontextuell auf der *Visualization*-Schicht befindet über einen Webservice bestimmte Messwerte von den Rechenkernen anfordern. Es obliegt dem Dashboard die passende Visualisierung in Form von Diagrammen oder Tabellen zu erzeugen. Wie in der Abbildung angedeutet können auch andere Systeme auf die *Calculation-and-Storage*-Schicht zugreifen, um die Messwerte weiterzuverarbeiten.

Die *Operation*-Schicht enthält Werkzeuge für den Betrieb und zur Überwachung der kompletten Mess-Infrastruktur (vgl. [67]). Für weitere Informationen und Details zu EMI und EMDB siehe Steffens [51] und Vianden [59].

### 3.3.1. Webservice in MeDIC

Dem Leitgedanken des Entwurfsmusters *Separation of Concerns* (SoC) [30] folgend sind in MeDIC das MeDIC-Cockpit, der Datenbus und die Rechenkerne in eigene Systeme unterteilt (siehe Abschnitt 3.3). Diese lose Kopplung ermöglicht es die Systeme weitgehend unabhängig voneinander zu entwickeln, zu erweitern und zu warten. Beispielsweise hat eine Änderung im EMDB-System durch die damit verbundene Neukompilierung, Testung und Bereitstellung eine Ausfallzeit zur Folge, die sich nicht direkt auf das MeDIC-Cockpit oder andere Konsumenten auswirkt. Die Systeme, welche den Webservice in Anspruch nehmen, bleiben bis zu einem gewissen Grad weiter funktionsfähig und können autonom weiterlaufen. Da der Zugriff auf aktuelle Messdaten unterbrochen ist, wäre im konkreten Fall des MeDIC-Cockpits ein Caching des letzten Zustands der Dashboard-Items denkbar und somit Autonomität sichergestellt.

Für die Trennung der Zuständigkeiten spricht in diesem Kontext auch die Dauer des Erstellungsprozesses (Build-Dauer). Systeme dieser Größenordnung brauchen, je nach Konfiguration der Entwicklungsumgebung und der verfügbaren Rechenleistung, mehrere Minuten um alle Phasen des Erstellungsprozesses zu durchlaufen[4]. Generell gilt, dass Build-Dauer und Systemgröße positiv proportional zusammenhängen. Auf Systemebene findet üblicherweise eine weitere Aufteilung in Komponenten statt, sodass diese vorab einzeln erstellt und im Erstellungsprozess des Gesamtsystems wiederverwendet werden können.

[4]Beim MeDIC-Cockpit sprechen wir von 3 bis 6 Minuten.

Wie zuvor schon angedeutet, bringt die Kapselung der Funktionalität der Rechenkerne als Webservice einen weiteren wichtigen Vorteil mit sich: die Wiederverwendung über Plattformgrenzen hinaus! Für eine erfolgreiche Kommunikation zwischen Dienstleister (Webservice) und Konsument (Client) müssen sich beide Seiten auf ein bestimmtes Protokoll und Datenformat einigen. In der Regel gibt der Webservice diese Schnittstellenbeschreibung vor.

Die wichtigsten Protokolle und Datenaustauschformate für Webservices wurden im Kapitel 2 vorgestellt.

## 3.4. Fehlende Variabilität

Die Rechenkerne lassen eine wichtige Eigenschaft vermissen: **Variabilität**. Diese würde es ermöglichen Einzelheiten in der Metrik-Berechnung von der Visualization-Schicht aus zu beeinflussen.

Dies soll anhand des folgenden Beispiels deutlich gemacht werden (angelehnt an [59]). Sei angenommen, ein Rechenkern zählt alle offenen Fehler-Tickets eines Projekts. Dieser Rechenkern stellt die Datenbasis für eine Visualisierung in Form eines Dashboard-Items im Dashboard dar. Angesichts des bevorstehenden Abgabetermins der Software möchte der Projektleiter das Dashboard-Item so modifizieren, dass nicht alle offenen Fehler-Tickets betrachtet werden, sondern lediglich solche mit höchster und zweithöchster Priorität. In der aktuellen Architektur müsste für jede Variante dieser Art ein eigenständiger Rechenkern angelegt werden, was schnell zu einer Explosion an Rechenkernen führen kann. Daraus würde sich sogleich ein erhöhter Wartungs- und Betriebsaufwand ergeben.

Alleine aus den Eigenschaften von Fehler-Tickets ergeben sich bereits vielfältige Kombinationsmöglichkeiten, die für eine Zählmetrik[5] relevant sein könnten (wie z.B. Status, Resolution, Type, Component, Keywords, Priority, Milestone, Comments). Natürlich sind nicht alle möglichen Kombinationen beziehungsweise Varianten sinnvoll. Manche stellen gar eine gänzlich neue Metrik dar.

[5] Eine Zählmetrik zählt Einheiten (hier Tickets) unter Berücksichtigung von Filterkriterien.