ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Dokumentation einer konzeptuellen Ausarbeitung eines Condition Monitoring Systems des Moduls Software-Architektur

vorgelegt von

Alexander Huber

Manh-Khang Richard Mai

Hendrik Haas

betreut und begutachtet von

Prof. Dr. Markus Esch

Saarbrücken, 22. März 2020

Inhaltsverzeichnis

1 Einleitung

Seit Jahrzehnten versuchen die Menschen ihre maschinellen Systeme immer effizienter und sicherer zu gestalten. Durch das Aufkommen von Computern besteht seitdem die Möglichkeit diese Systeme digital und nutzerfreundlich zu überwachen. So nehmen sie sensorische Werte der Anlage auf und verarbeiten diese. Es können so beispielsweise Abschätzungen mittels linearer Regression über die Lebensdauer einer solchen Anlage erfasst und diese dann in einem Diagramm dargestellt werden. Solche Systeme sind und werden in Zukunft noch viel wichtiger werden, da hier statistische Methoden eingesetzt werden können, die für den modernen Fertigungs- und Wartungsprozess unerlässlich sind.

1.1 Problemstellung

Die Firma HYDAC Systems und Services GmbH entwickelt derzeit ein System zum erfassen, verarbeiten und auswerten von Messwerten und Prozessdaten in Echtzeit. Allgemein ist solch ein System auch unter dem Begriff Condition Monitoring bekannt und ermöglicht es Prozesse zu überwachen, zu planen und gegebenenfalls zu optimieren. Es wird demnach zur Prozessoptimierung verwendet und erhöht die Verfügbarkeit von Anlagen. Im Rahmen der Machbarkeit und Untersuchung unterschiedlicher Herangehensweisen, ist es für die Firma interessant wie Studenten der Hochschule htwsaar ein Condition Monitoring System anhand gegebener Anforderungen konzipieren würden. Folgende Anforderungen verdeutlichen die zentrale fachliche Aufgabenstellung:

- Vordefinierte Datensätze sind aus einer Datenquelle gefiltert oder ungefiltert zu empfangen und in einer Datenbank persistent abzuspeichern. Dabei kann diese Datenquelle verschiedene Protokolle sein, wie HFI-MM, HFI-CM, Modbus TCP/RTU, CAN/CANopen, S7 protocol, OPC-UA (client and server), HTTP und WebSocket
- Das System benötigt eine Benutzerverwaltung um Nutzern und Gruppen Zugriffsrechte auf dem System zu ermöglichen. Die Zugriffsrechte sind lesend, schreibend und administrativ. Solange man administrative Rechte hat kann man die Rechte anderen Nutzern und Gruppen verteilen.
- Es wird ein Web-Interface benötigt um auf die einzelnen Komponenten zugreifen zu können. So soll es ein Dashboard geben, das individuell angepasst werden kann und beispielsweise Prozesse anzeigt.

1.2 Zielsetzung

Das Ziel dieser Arbeit soll es sein, einen Architekturentwurf für ein Condtion Monitoring auszuarbeiten, welcher unter Anwendung moderner Architekturansätze zu gestalten ist. Die folgenden primären Qualitätsziele sind dabei genau zu beachten:

- Effizienz (Performance | Leistung): Das System soll ca. 500 Datensätze innerhalb einer Sekunde aus ein oder mehreren Datenquellen erfassen können.

- Verfügbarkeit (Erreichbarkeit): Das System soll eine Hochverfügbarkeit von mindestens 99,9% aufweisen und darf demnach maximal 8h 45min im Jahr ausfallen.
- Nutzbarkeit (UX): Das User-Interface soll einfach zu verstehen sein und einem Bediener innerhalb einer Stunde erlernbar sein. Des weiteren soll die Oberfläche für den Benutzer individuell anpassbar sein.

1.3 Stakeholder

Die Stakeholder werden durch eine Stakeholderanalyse identifiziert und bezeichnen die für das Projekt relevanten Personengruppen. Hierzu ist festzuhalten, welchen Vorteil die jeweiligen Gruppen haben und welche individuellen Erwartungen an das System gestellt werden. Durch eine Marktanalyse in Betrachtung des Einsatzumfelds und anhand der Einsatzmöglichkeiten haben sich uns folgende Personengruppen als Stakeholder an das System herausgestellt:

- Fach- und Führungskräfte
- Anwendungs- und Testentwickler
- Linien- und Produktionskräfte
- Systemadministratoren

Die Fach- und Führungskräfte sind insbesondere an einer zielführenden Übersicht und einer Aggregation von planbaren Artefakten interessiert, welches ihnen wiederum bei der Unterstützung und Planung des Arbeitsprozesses hilft. Die Anwendungs- und Testentwickler erwarten einen niedrigen Integrationsaufwand bei Implementierung neuer Funktionalitäten. Die Linien- und Produktionskräfte erwarten eine einfache Ansicht der Zustandserfassung, sodass sie das System einfach überwachen können. Die Systemadministratoren erwarten, dass das System eine sofortige Informationsmitteilung und eine einfache Maßnahmenkontrolle bereitstellt, damit einerseits die Systeme auf Fehler überwacht und reagiert werden können.

1.4 Gliederung

Das erste Kapitel gibt einen Überblick über die Themen, welche folgend in der Dokumentation ausformuliert sind. Es schafft Klarheit darüber wer, warum und wozu solch ein System benötigt wird und definiert die zentralen Merkmale. Im Kapitel 2, wird dann auf Randbedingungen eingegangen, die nicht direkt in den Anforderungen beschreiben sind, welche aber trotzdem bei einer Entwicklung beachtet werden sollen. Das 5 befasst sich danach mit den gestellten Anforderungen und beschreibt ein mögliches Konzept nach Kriterien. Kapitel Im letzten Kapitel 6 werden wir das Vorgehen abschließend beurteilen und einen Ausblick wagen.

2 Einflussfaktoren

2.1 Organisatorische Einflussfaktoren

Tabelle 2.1: Organisatorische Einflussfaktoren

Organisatorische Einflussfaktoren	Felxibilität und Veränderbarkeit	Einfluss
Das Projekt ist innerhalb von 3 Jahren abzuschließen	Eine Planabweichung ist nicht zulässig	kleines Projekt mit angegebener Hardware
Kostengünstig	35.000 €	komplexere Software
Wartungsperson bzw. Administration	1-2 Personen	Für Fehlerkorrekturen und Wartungen

2.2 Technische Einflussfaktoren

Tabelle 2.2: Technische Einflussfaktoren

Technische Einflussfaktoren	Felxibilität und Veränderbarkeit	Einfluss
Das Edge-Device kommuniziert über unterschiedliche Bussysteme mit verschiedene Geräten	Die eingesetzten Busstechnologien kann sich alle paar Jahre ändern	Kapselung der Kommunikation in einer Komponente
Geräte sollen möglichst 24/7 aktiv sein	Hotfixes und Wartungen	Wenn eines der Geräte ausfällt
Stale für die GUI	Nicht zu viel an Design aufwenden	Könnte grafische Ergebnisse anzeigen und/oder mehr Benutzerfreundlich

3 Kontextabgrenzung

In der Kontextabgrenzung grenzen wir alle Kommunikationspartner vom System ab und stellen somit unsere externen Schnittstellen fest. Dazu wird der Kontext einerseits fachlich, als auch technisch voneinander abgegrenzt.

3.1 Fachlicher Kontext

Die fachliche Kontextabgrenzung dient uns dazu alle Kommunikationspartner zusammen mit den jeweiligen ein und Ausgabedaten mit dem System zu erläutern. Folgende Abbildung veranschaulicht die fachliche Kontextabgrenzung:

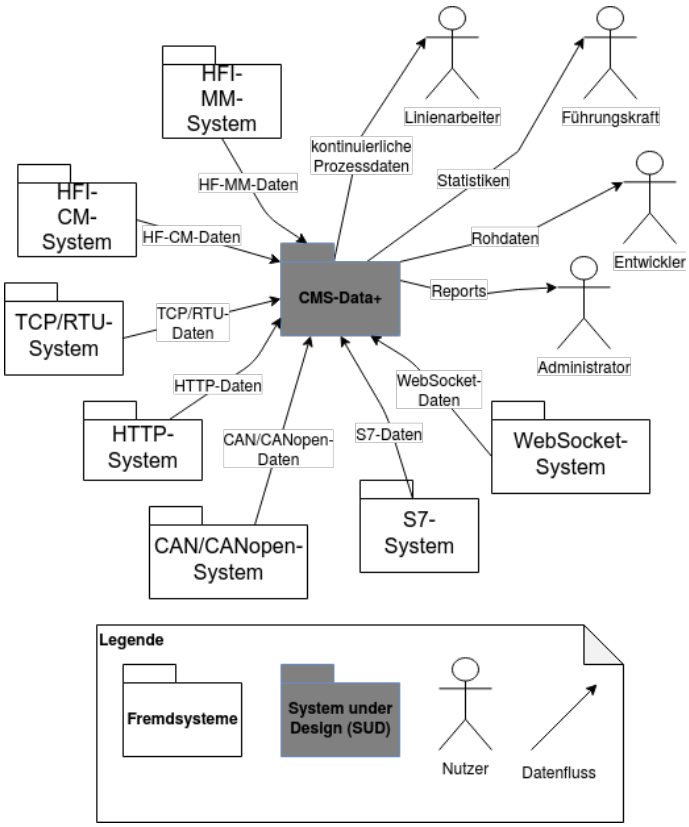

Abbildung 3.1: Fachliche Kontextabgrenzung

Wie das Diagramm veranschaulicht, dient das Data+ als Datensenke für alle angezeigten Systeme. Jedes dieser Systeme kann über eine eigene Schnittstelle bzw. Protokoll verfügen und über jenes die protokollspezifischen Daten an das Data+ senden. Folgende Tabelle stellt noch einmal die Kommunikationspartner und den fachlichen Kontext zum System dar:

Kommunikations-partner	Eingabe	Ausgabe
HFI-MM-System	Sensordaten	HFI-MM-Protokolldaten nach HFI-MM Standard
HFI-CM-System	Sensordaten	HFI-CM-Protokolldaten nach HFI-CM Standard
TCP/RTU-System	Sensordaten	Protokolldaten nach TCP/RTU Standard
HTTP- System	4 verschiedene Arten von Daten(User, Group, Station, Datapoint)	Daten im JSON-Format
CAN/CANopen-System	Sensordaten	CAN/CANopen-Protokolldaten nach Standard
S7-System	Sensordaten	S7-Protokolldaten nach Standard
WebSocket-System	Streamdaten	Daten im ProtoBuf Format

Tabelle 3.1: Fachliche Kontextabgrenzung der Kommunikationspartner

Alle einzelnen Kommunikationspartner haben selbst wiederum Eingangsdaten und geben alle Daten aus verschiedenen Protokollen dem SUD weiter. Dieses verarbeitet die eingehenden Daten und wertet diese aus.

3.2 Technischer Kontext

Der technische Kontext definiert die Kanäle und das Übertragungsmedium über dem unser System mit externen Komponenten interagiert. Dabei wird erklärt, wie und über welche technischen Kanäle unsere fachlichen Ein- und Ausgaben ablaufen. Zur Veranschaulichung wird dies im folgenden UML Deployment-Diagramm dargestellt:

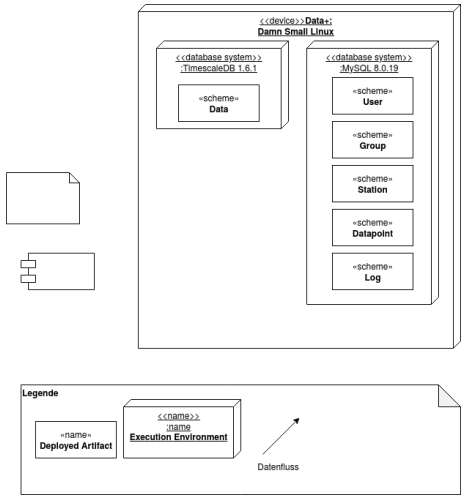

Abbildung 3.2: Technische Kontextabgrenzung

4 Konzept

In diesem Kapitel wird nun der Entwurf vorgestellt. Dazu möchten wir als erstes die mögliche Lösungsstrategie unseres Systems anschauen und vertiefen warum wir die nötigen Entscheidungen getroffen haben. Dann werden verschiedene Sichten auf das System gezeigt, die das Veranschaulichen sollen.

4.1 Lösungsstrategie

In diesem Abschnitt werden nun die grundlegenden Entscheidungen und Lösungsansätze unseres Entwurfs vorgestellt. Weiteres Vorgehen in Hinblick auf Implementierung hängt hiervon ab. In den folgenden Technologieentscheidungen werden die Entscheidungen der Architektur festgelegt.

4.1.1 Datenbank

Die Datenbankkomponente ist mit das Herzstück eines Condition Monitoring Systems. Systeme die lange Speicherzeiten benötigen sind den Anforderungen ungenügend. Da das System eine möglichst hohe Lese- und Schreiblast vorweisen muss, bieten sich Extensible-Record Stores Datenbanken an. Diese sind Erweiterungen der Key-Value Stores indem sie für den Wert eine Struktur (ähnlich dem Schema einer Tabelle) zulassen. Schlüssel sind als Bytearray und der Wert besteht aus Paaren von Spaltenname und -wert. Im Gegensatz zum relationalen Modell gibt es kein fixes Schema. Ein solches System ist zum Beispiel Hypertable, das mit einem geringen Speicherplatzbedarf sich optimal eignet. Nach eigenen Angaben soll das System sich sogar um die Konsistenz der Daten kümmern, während sich viele NoSQL-Datenbanken mit einer vermeintlichen Konsistenz der Daten begnügen. Daher haben wir eine kleine Auswahl an Datenbanksystemen miteinander verglichen. Eine Grundeigenschaft war, dass das System ohne Java auskommt, weshalb Cassandra und HBase wegfallen.

4.1.1.1 Hypertable

Das im März 2016 eingestellte Projekt Hypertable basiert auf BigTable und verteilten Daten. Geschrieben wurde es in C++ und ist ein schemafreies System. Die Vorteile liegen in der sofortigen Konsistenz der Daten und der Möglichkeit gleichzeitig lesen und schreiben zu können. Der große Nachteil ist, dass es keine Updates mehr gibt, da das Projekt nicht fortgesetzt wird.

4.1.1.2 Orcale Berkeley DB

Orcale Berkeley DB gibt es schon seit 1994 und wurde je nach Edition in C, C++ oder Java implementiert. Es handelt sich dabei um einen Key-Value Speicher, der XML-Unterstützung bietet, SQL und sekundäre Indexierung kann. Die Datenbank wird nach dem ACID-Prinzip gefüllt und ist trotzdem noch sehr schnell bzw. und mit der richtigen Konfiguration auch komplett parallel nutzbar.

4.1.1.3 Redis

Redis ist eine beliebte Datenbanksoftware, die seit 2009 auf dem Markt ist. Sie ist ein in C geschriebener Key-Value-Speicher, der eine große, aber keine vollständige Konsistenz der Daten verspricht. Dafür ist der zugriff komplett parallel und kann durch einige Einstellungen auch stark konsistent werden.
TODO Welches nehmen wir und warum?!?!!?! TODO!11!1!!! Benchmark: Klick

4.1.2 Serialisierung Capn Proto

Da wir mit einem System arbeiten, das eingehende wie ausgehende Daten, in dem System fachterminologisch als Datenpunkt bezeichnet, verarbeitet und mit programminternen sowie /-externen Einheiten kommuniziert, muss bekanntlich auch eine Serialisierung dieser Daten erfolgen. Bei einer Reihe zur Verfügung stehender Serialisierungstechnologien ist es für das Produkt nötig die Vor- und Nachteile der jeweils einzelnen abzuwägen. Da es für das System von großer Wichtigkeit ist mit einem möglichst großen Datendurchsatz performant umgehen zu können, können textbasierte Serialisierungstechniken wie JSON, XML, YAML usw. kategorisch abgelehnt werden. Der Grund dafür ist, das im allgemeinen ein zu großer Speicherbedarf benötigt wird, was zur Folge hat, dass die Serialisierungszeit und damit auch die Laufzeit stark ansteigt.
Eine Möglichkeit die Laufzeit zu verringern wäre eine adhoc single String Technik zu verwenden. Diese muss jedoch redundant für jede Sprache geschrieben werden, was uns in der Portierbarkeit einschränken würde.
Die nach unserer Meinung nach beste Lösung für dieses Problem wäre es demzufolge eine schemagesteuerte binäre Serialisierungstechnik zu verwenden. Zum Einsatz soll Capn Proto verwendet werden. Durch dessen flexible Schemadefinition lässt sich im Vergleich zu den anderen Portierbarkeit erhöhen, da dieses Framework den Serialisierungscode anhand der Schemadefinition selbst erzeugt. Die binäre Darstellung der Daten erlaubt zudem eine deutlich effizientere Übertragung der Daten. Capn Proto ist im Vergleich zu Protobuf fast doppelt so schnell, siehe Proto Benchmarks. Die Bibliotheken sind einfach zu nutzen und in C++ geschrieben.

4.1.3 Parallelisierung

Damit alle Systeme und Komponenten zeitgleich ihre Aufgaben erfüllen können, wird die Software in Micro Services aufgebaut, die unabhängig voneinander parallel arbeiten können.
Weitere Vorteile liegen in der unabhängigen Entwicklung, Implementierung und Wartung der einzelnen Bausteine. Allerdings wird dadurch auch ein Load Balancer nötig, der die Ressourcen der Hardware nach den Anfragen auf die einzelnen Prozesse verteilt.

4.1.4 Fehlererkennung

Pings oder ICMP Request/Replies werden zwar oft genutzt um zu sehen, ob ein Server noch online ist, aber da ICMP betriebssystemspezifisch implementiert ist und in manchen Sicherheitsrichtlinien deaktiviert sein muss, kann man das hier nicht für eine dauerhafte Taktik verwenden. Außerdem erhält man keine Einsicht darüber, ob ein Programm auf der Maschine ausgeführt ist, was in unserem Fall aber wichtig wäre.
Mit Hilfe von Monitoring kann man die Antwortzeiten und die Uptimes von allen Anwendungen eines Servers abfragen und darstellen. Diese Methode ist zielführender, braucht allerdings eine zusätzliche Schnittstelle, die nicht immer bereitgestellt werden kann. Auch

verbraucht sie einige Ressourcen und die Abfrage kann bzw. sollte nicht in zu kurzen Zeitintervallen stattfinden.
Heartbeat ist eine periodischer Nachrichtenaustausch zwischen einem Prozess und einem Kontrollsystem. Da unsere Anwendung in Echtzeit große Datenmengen bewältigen muss, ist es wichtig, dass der Heartbeat schnell und einfach gesendet und empfangen wird um keine anderen Daten/Datenverarbeitung zu behindern. Mit Hilfe von Condition Monitoring kann man zum Beispiel bei einer Maschine aus Sensordaten Schwingungen und Temperaturen erfassen, die evtl. für Ausfälle sorgen. Um das bei unserem Server anzuwenden, muss man die CPU-Temperatur überwachen und ggf. den Serverschrank weiter runterkühlen um Ausfälle zu vermeiden. Voting ist nicht nur prozessorlastig, sondern wird am besten auch auf mehreren Geräten gleichzeitig ausgeführt was langsam und kostenintensiv ist, daher verwerfen wir diese Möglichkeit.
Zeitstempel kann man einfach in die Datenbank einpflegen um evtl. fehlerlastige Zeiten herauszufinden und zu löschen oder zu korrigieren. Eine größere Validierung wird wieder zu CPU-lastig und kann daher nicht direkt beim Speichern der Daten durchgeführt werden. Aus Kostengründen müssen wir auch alle Wiederherstellungsmaßnahmen, die Redundanz beinhalten leider ausschließen, da das Budget zu klein gesetzt wurde. Daraus folgt treten Softwarefehler ein, muss man mit Hilfe von Hotfixes und bei Hardwarefehlern mit einem Austauschgerät arbeiten. Außerdem kann man bei Exceptions, die zur Laufzeit auftreten den jeweiligen Stacktrace mit der dazugehörigen Eingabe an eine Kontrollstelle abspeichern. Sobald eine kritische Anzahl an Exceptions pro Minute/Stunde auftritt muss der Fehler in der Software gepatcht werden. Im Softwaredesignprozess muss dabei darauf geachtet werden, dass Exceptions immer gehandelt werden und das auch in sehr ungewöhnlichen oder gar unmöglichen Fällen um in jedem Szenario handlungsfähig zu bleiben.
Abschließend haben wir uns für eine Kombination aus Condition Monitoring, Exception Handling und Zeitstempel entschieden, weil diese kaum Leistung benötigen und Kosten verursachen. Hotfixes für Ausfälle sind gerade im Pilotbetrieb erstmal günstiger als redundante Hardware und komplexe Monitoringtools. Diese können bei Bedarf in einer späteren Variante auf besserer Hardware eingebaut werden.

4.2 Bausteinsicht

4.3 Laufzeitsicht

4.4 Verteilungssicht

5 Qualitaetsszenarien

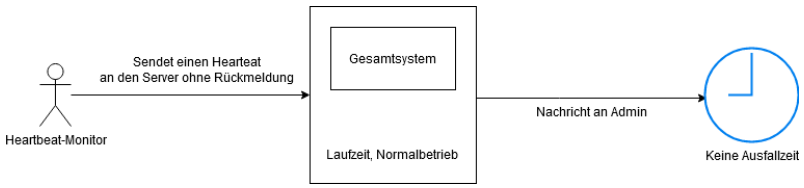

Abbildung 5.1: Verfuegbarkeits-Szenario

Verfuegbarkeits-Szenario
Source of Stimulus: Heartbeat Monitor
Stimulus: Sendet einen Heartbeat an den Server ohne Rückmeldung
Environment: Laufzeit, Normalbetrieb
Component: Gesamtsystem
Response: Nachricht an Admin
Response Measure: Keine Ausfallzeit

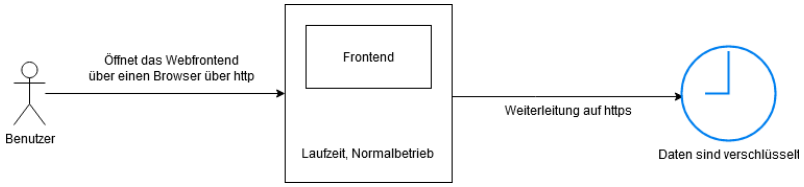

Abbildung 5.2: Effizienz-Szenario

Sicherheits-Szenario
Source of Stimulus: Benutzer
Stimulus: Öffnet das Webfrontend über einen Browser über http
Environment: Laufzeit, Normalbetrieb
Component: Frontend
Response: Weiterleitung auf https
Response Measure: Daten sind verschlüsselt

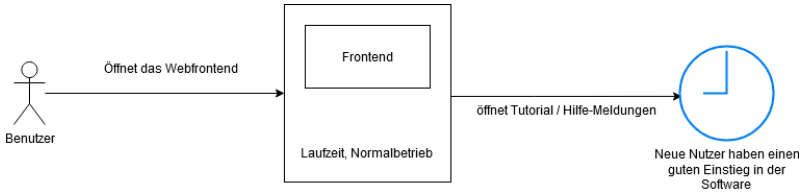

Abbildung 5.3: Nutzbarkeit-Szenario

Nutzbarkeits-Szenario Source of Stimulus: Benutzer
Stimulus: Öffnet das Webfrontend
Environment: Laufzeit, Normalbetrieb

Component: Frontend
Response: Öffnet das Tutorial / Hilfe-Meldung
Response Measure: Neue Nutzer haben einen guten Einstieg in der Software

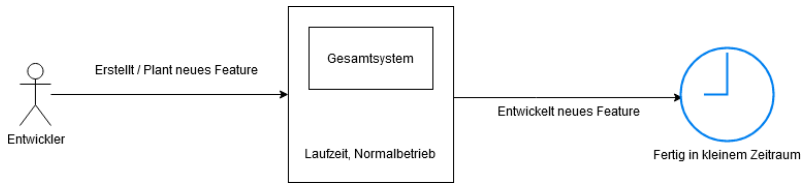

Abbildung 5.4: Erweiterbarkeits-Szenario

Erweiterbarkeits-Szenario
Source of Stimulus: Entwickler
Stimulus: Erstellt / Plant neues Feature
Environment: Laufzeit, Normalbetrieb
Component: Gesamtsystem
Response: Implementierung ohne negativen Sideeffects
Response Measure: Implementierung innerhalb einer Woche

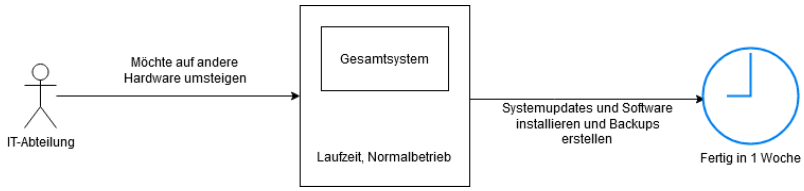

Abbildung 5.5: Portierbarkeit-Szenario

Portierbarkeits-Szenario
Source of Stimulus: IT-Abteilung
Stimulus: Möchte auf andere Hardware umsteigen
Environment: Laufzeit, Normalbetrieb
Component: Gesamtsystem
Response: Systemupdates und Software installieren und Backups erstellen
Response Measure: Fertig in 1 Woche

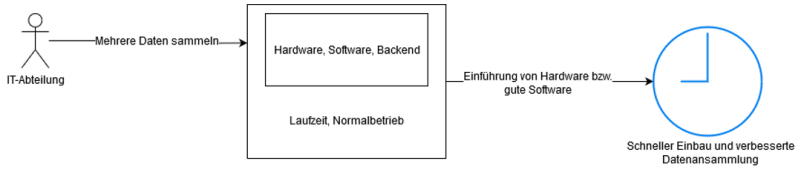

Abbildung 5.6: Effizienz-Szenario

Effizienz-Szenario
Source of Stimulus: Entwickler
Stimulus: Mehrere Daten sammeln
Environment:Laufzeit, Normalbetrieb
Component: Hardware, Software, Backend
Response: Einführung von Hardware, bzw. gute Software um Datenpunkte zu sammeln

Response Measure: Schneller Einbau und verbesserte Datensammlung

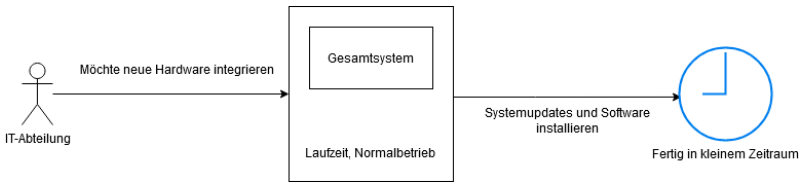

Abbildung 5.7: Hardware-Szenario

Hardware-Szenario
Source of Stimulus: IT-Abteilung
Stimulus: Möchte neue Hardware integrieren
Environment:Lautzeit, Normalbetrieb
Component: Gesamtsystem
Response: Systemupdates und Software installieren
Response Measure: Fertig in kleinen Zeitraum

6 Fazit

Literatur

[1] Donald E. Knuth. „Computer Programming as an Art“. In: *Communications of the ACM* 17.12 (1974), S. 667–673.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

Anhang

A Erster Abschnitt des Anhangs

In den Anhang gehören „Hintergrundinformationen“, also weiterführende Information, ausführliche Listings, Graphen, Diagramme oder Tabellen, die den Haupttext mit detaillierten Informationen ergänzen.

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.