Inhaltsverzeichnis

1 Von der Aufgabe zur Anwendung 2
1.1 Aufgabenstellung....2
1.2 Analyse....4
1.2.1 Struktur....5
1.2.2 Aufbau....6
1.2.3 Projektbeschreibung 8
1.2.4 Abgrenzung 8
1.2.5 Vorgaben....9
1.2.6 Gesetze und Normen....9
1.2.7 Rückfragen 9
1.2.8 Algorithmen....10
1.2.9 Vorgänge / Verfahren....11
1.2.10 Einschränkungen....11
1.2.11 Fehler 12
1.2.12 Form 12
2 Entwurf....13
2.1 Strukturierung des Entwurfs....14
2.2 Vorteile einer Strukturierung des Entwurfs....16
2.2.1 Prototyping und Tests....17
2.2.1.1 Top-Down....21
2.2.1.2 Priorisierung sowie Bottom-up....23
2.2.1.3 Meilensteine....24
2.2.1.4 Erweiterung eines Entwurfs....25
2.2.1.5 Spezialisierung....28
2.2.1.6 Schnittstellen....29
3 Erstellung eines Entwurfs 30
3.1 Grobentwurf....30
3.2 Feinentwurf....31
3.3 Spezialisierung....34
3.4 Struktur einer Anwendung....35
3.5 Abgeleitete Anwendungen....38
4 Beispielaufgabe....39
4.1 Aufgabenstellung....39
4.2 Überlegungen....39
4.3 Kontext des Auftraggebers 42
4.4 Aufgabenstellung....42
4.5 Analyse....42

1 Von der Aufgabe zur Anwendung

Früher, als die Softwareprodukte noch klein waren, war es nicht selten üblich, dass Software einfach mal so entstanden ist. Der Programmierer sah das Problem und fing an zu tippen ohne jegliche Gedanken daran, wie das Ergebnis seiner Arbeit später aussehen soll. Auch wenn dieser Ansatz in der Industrie stellenweise immer noch vorhanden ist, hat es sich herausgestellt, dass dies keine zufriedenstellende Vorgehensweise ist, da auf diese Art und Weise erstellte Software weder wartbar, noch zukunftsfähig ist, so dass beim Versuch die alte Software den neuen Gegebenheiten anzupassen öfters eine komplette Neuentwicklung deutlich günstiger ausfällt. Dieses Dokument will versuchen dem Leser ein paar Techniken zu vermitteln wie dieses Problem mit einem richtigen Vorgehensmodell umgangen werden kann. Dieses Modell gliedert sich in folgende Kernprozesse:

- Planung (wird aus Zeitgründen in diesem Dokument nur oberflächlich behandelt)
- Analyse
- Entwurf
- Implementierung
- Qualitätssicherung /Tests (wird aus Zeitgründen in diesem Dokument nur oberflächlich behandelt)

Dabei sei angemerkt, dass die Übergänge zwischen den einzelnen Prozessen nicht immer genau abgegrenzt werden können so dass z.B. die Entscheidung ein bestimmtes Framework zu benutzen zu der Analyse aber auch zum Entwurf gehören kann. Die Grenze, die an dieser Stelle gezogen werden könnte, wäre, dass die Entscheidung ein Framework zu benutzen zu der Analyse gehört, aber die Entscheidung ein bestimmtes Framework zu verwenden eine Angelegenheit des Entwurfs ist.

1.1 Aufgabenstellung

Die Aufgabenstellung dient dem Auftraggeber zur Festlegung des Umfangs der Arbeiten die für die Lösung seines Problems notwendig sind. Dies wird, aus verständlichen Gründen, fast ausschließlich aus der Sicht des Auftraggebers oder des späteren Anwenders gesehen und nicht aus der Sicht desjenigen der zu diesem Problem eine Lösung in Form eines Softwarepakets entwickeln soll. Der Grund dafür liegt in der Regel in fehlendem Fachwissen um es technisch zu beschreiben und anderseits sind für den Anwender die Eigenschaften der Eingabemaske wichtiger und deutlich „greifbarer“ als irgendwelche Algorithmen. Deswegen wird in der Aufgabenstellung beschrieben, was der Benutzer am Schluss sehen, bzw. welche Probleme oder Problemklassen er mit der zu erstellenden Anwendung lösen will, nicht der Weg, den der Entwickler nehmen muss um die Anwendung zu implementieren. Es ist auch sehr zu beachten aus welchem Kontext die Aufgabenstellung entstanden ist, bzw. die Umgebung in der die Applikation später verwendet wird. Es ist vor allem deswegen von einer großen Bedeutung, da diese „versteckte Anforderungen“ oder bestimmte sonstige Vorgaben beinhalten kann, die auf Grund des vorhandenen Fachwissens für den Aufgabensteller offensichtlich, für den Auftragsempfänger aber vollkommen fremd sind.

Ein weiterer Aspekt könnte die verwendete Fachsprache des Auftraggebers sein, so dass manche Wörter oder Ausdrücke eine ganz andere Bedeutung für den Auftraggeber als für den Auftragnehmer haben. Als Beispiel kann hier das allmorgendliche „öffnen“ des Computers durch die Büroangestellten dienen, die durch das Drücken der Starttaste bewerkstelligt wird (Vergl. „Öffnen eines Ordners“), sowie das gegenseitige „Appen“ der älteren Generationen (oder „Ich schicke Dir eine App) was nichts anderes heißt als eine Nachricht zu verschicken (von dem Nachrichtendienst „WhatsApp“).

Was auch noch wichtig ist, ist der Umstand wie eine Aufgabenstellung entstanden ist. Bei größeren Projekten ist diese in der Regel nicht von einer einzigen Person verfasst, sondern einzelne Teilanforderungen stammen oft von Mitarbeitern der betroffenen Abteilungen (Abbildung 1). Dies kann dazu führen, dass die in der Aufgabenstellung erfassten Angaben redundant, inkonsistent widersprüchlich usw. sind. Des Weiteren ist es möglich, dass bei der Vereinheitlichung des Textes durch den Editor, der nicht immer ein notwendiges Fachwissen auf allen Gebieten haben kann, es zu Verfälschungen oder zur Informationsverlust kommt. Aus den oben genannten Gründen ist es eigentlich unumgänglich bei größeren Projekten vor der Analyse ein Pflichten- und Lastenheft zu erstellen (Planungsphase).

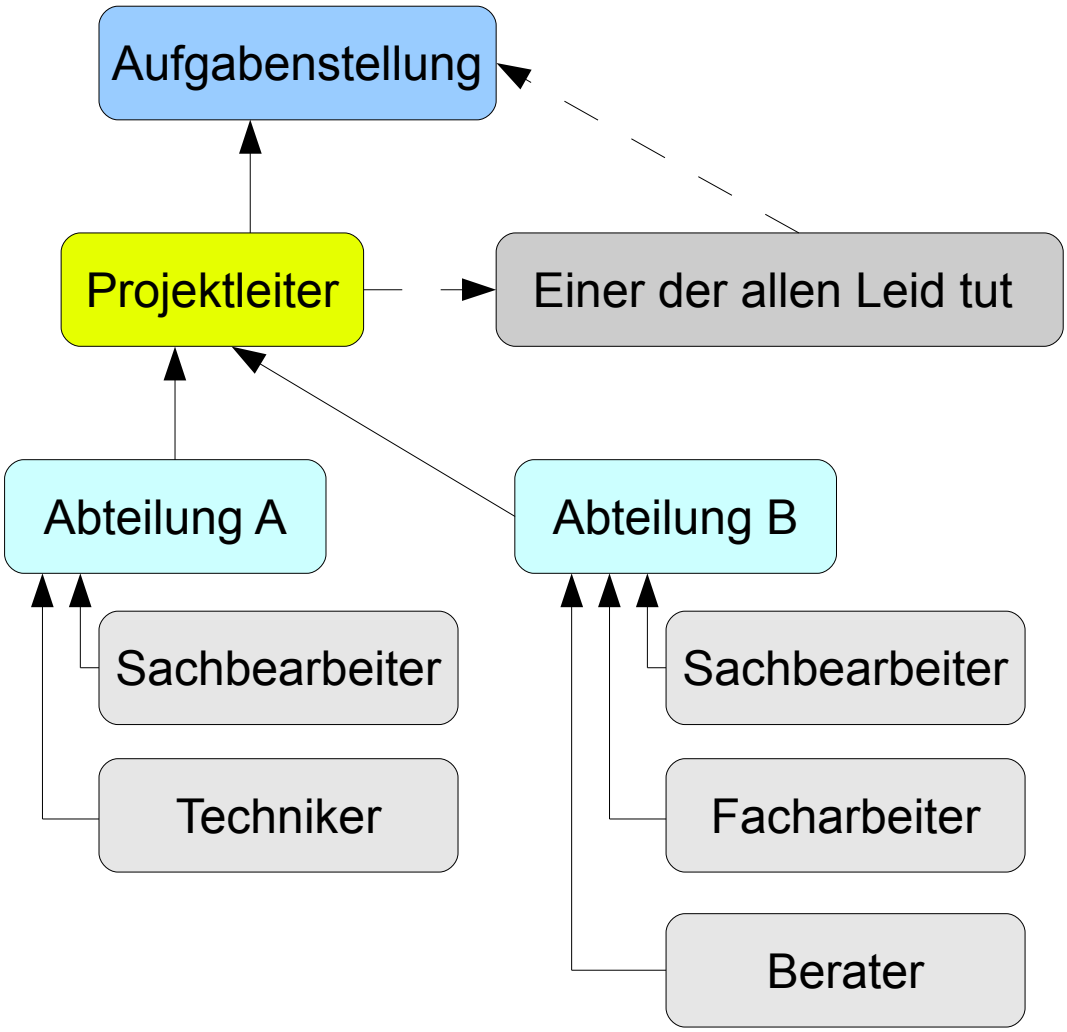

Abbildung 1: Entstehung einer Aufgabenstellung

1.2 Analyse

Die Analyse beschäftigt sich grundsätzlich mit der Frage was eigentlich im Projekt zu tun ist. Dabei ist zunächst darauf zu achten, dass die gegebene Aufgabenstellung vom Auftragnehmer richtig aufgefasst wurde. Im Zweifelsfall ist eine Konsultation mit dem Auftragsteller bezüglich der angenommenen Unstimmigkeiten durchzuführen. Es dürfen dabei keine Annahmen („er hatte damit wohl gemeint...“ in der Regel nämlich meistens nicht) vorgenommen werden, oder eigenhändige Änderungen („So ein Idiot, das geht doch gar nicht so“) gemacht werden. Da der Kunde unter Umständen noch Systeme verwendet, die eigentlich Relikte einer längst vergangenen Zeit darstellen (Abbildung 2), können diese eine bestimmte Vorgehensweise beim Projekt erzwingen. Anderseits ist es wohl möglich, dass es technisch bessere Lösungen gibt, als die vom Kunden ausgedachten. Soll es so sein, dann beschränkt sich allerdings der Entwickler auf Vorschläge wie es besser zu realisieren wäre. Auch alle sonstigen Ungereimtheiten werden durch Rücksprachen gelöst. Dabei ist es auch von Vorteil, wenn es möglich ist, sich das Gesamtsystem (auch Abläufe) des Kunden anzuschauen und z.B. die Mitarbeite, die später mit der Software arbeiten müssen, zu befragen. Noch besser ist es, die eigentlichen Geschäftsvorgänge einmal selbständig durchzuführen, um ein Gefühl dafür zu bekommen, wie das System arbeiten sollte. An dieser Stelle wird jetzt einer unserer Studenten erwähnt, der im Rahmen seiner Abschlussarbeit eine betriebswirtschaftliche Software entwickeln sollte, und in den ersten 4 Wochen seiner Thesis zuerst als Sachbearbeiter sich das notwendige Wissen über die Betriebsvorgänge aneignete. Der zweite Teil der Analyse besteht darin, den eigentlichen Umfang der Arbeiten, der zur Verwirklichung des Projektes notwendig ist, zu bestimmen. Dabei ist zu beachten ob z.B. Frameworks, Bibliotheken oder ältere Projekte bei der Lösung des Problems behilflich sein können.

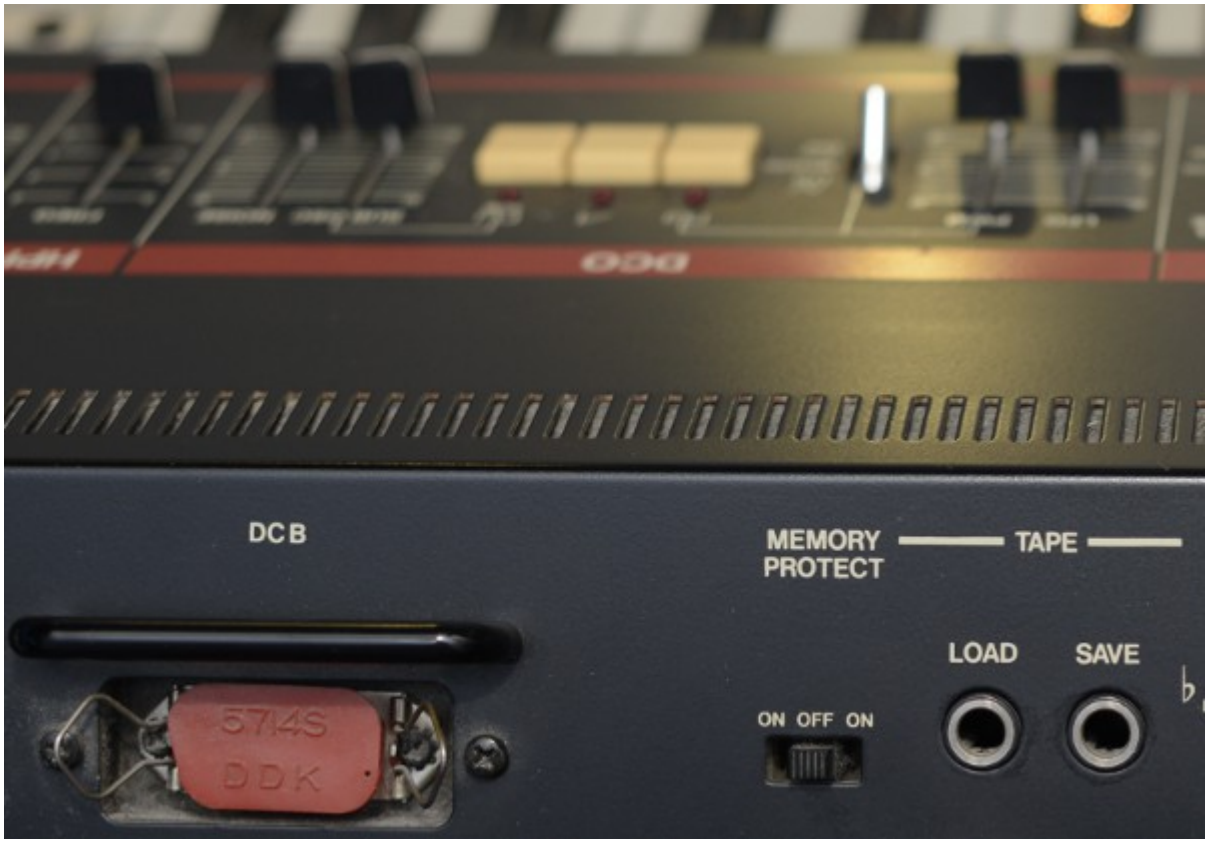

Abbildung 2 ehemals moderne Schnittstellen

1.2.1 Struktur

Da sich die meisten Projekte in Ihrer Struktur deutlich voneinander unterscheiden, ist es schwer eine Schablone für die Analyse vorzugeben. Eine Analyse, deren Ziel es ist herauszufinden, ob ein Projekt überhaupt durchführbar ist (Machbarkeitsstudie), unterscheidet sich deutlich von der Analyse eines Prototyps (proof of concept) das dann wiederum ganz anders aussieht wie ein Projekt, das produktiv bei einem Kunden installiert wird. Genauso wie der Zweck spielt auch die Größe eines Projektes eine entscheidende Rolle - die Analyse für die Funktionsweise eines Taschenrechners hat mit der einer Kernreaktorsteuerung nur noch den Namen gemeinsam. Im weiteren Verlauf dieses Dokuments sollen die wesentlichen Elemente einer Analyse skizzenhaft vorgestellt werden. Welche von diesen Elementen dann tatsächlich in der Analyse verwendet werden, hängt einerseits von dem Thema selbst, anderseits aber auch von der Erfahrung und dem literarischen Geschick des Verfassers, ab. Es ist auch nicht immer möglich eine klare Trennung zwischen diesen Elementen zu ziehen da sie entweder voneinander abhängen oder ineinander übergehen. Als Beispiel wäre hier ein Algorithmus zu nennen, der gewählt werden muss weil er als einziger einer Norm entspricht.

1.2.2 Aufbau

Der Aufbau der Analyse gleicht zuerst mal grob allen anderen wissenschaftlichen Dokumenten. Sie wird nach der Top-Down Methode aufgebaut (Siehe Kapitel 2.2.1.1). Das heißt, Informationen die das Gesamtprojekt betreffen werden für den Leser hervorgehoben indem diese zuerst aufgeschrieben werden. Zu diesen gehören:

- Der Projektname bzw. die kurze Beschreibung des Projekts
- Die an dem Projekt beteiligten Entitäten (Abteilungen, Firmen, Personen)
- Bei Teilprojekten Informationen über die an dem Gesamtprojekt beteiligten Entitäten
- Globale Vorgaben, die für das gesamte Projekt (-teil) eingehalten werden müssen

Danach können die weiteren Einzelheiten und Eigenschaften des Projekts bzw. die Ergebnisse der Analyse beschrieben werden. Auch hier soll nach Möglichkeit hierarchisch vorgegangen werden. Es sollte immer nur ein Aspekt der Analyse gleichzeitig behandelt werden. Falls dieser Abhängigkeiten oder Verbindungen zu anderen Aspekten hat, empfiehlt es sich an den gegebenen Stellen nicht zur Beschreibung des zweiten Aspekts „mitten drin“ überzugehen, sondern nur durch eine Erwähnung dieser Abhängigkeit eine „Brücke“ zu schaffen. Zum Beispiel:

a) Die über die <Maske> eingegebenen Daten sind auf dem Massenspeicher zu Speichern. Sollte eine SD Karte im System vorhanden sein, sind Daten auf der Speicherkarte zu speichern.

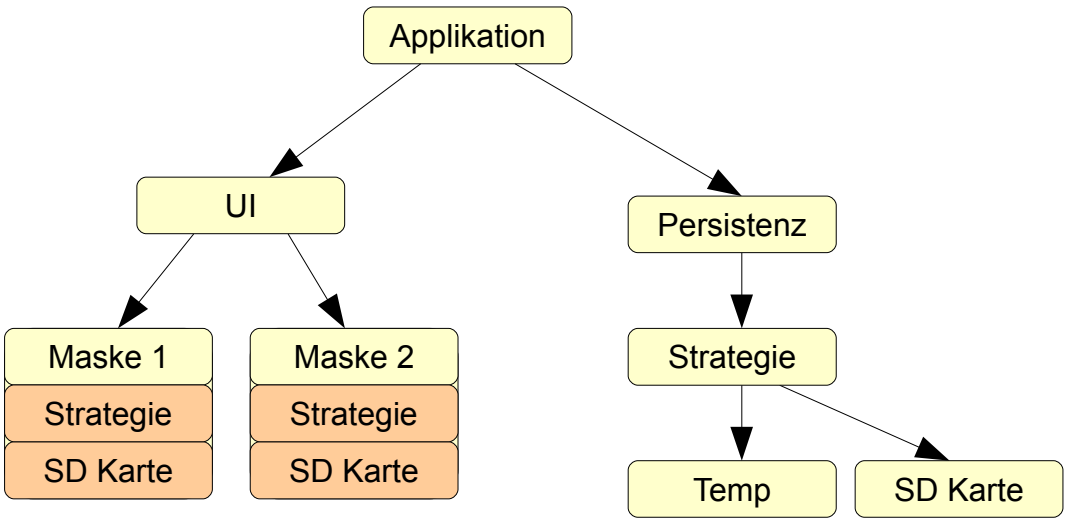

Abbildung 3: Struktur einer redundanten Analyse

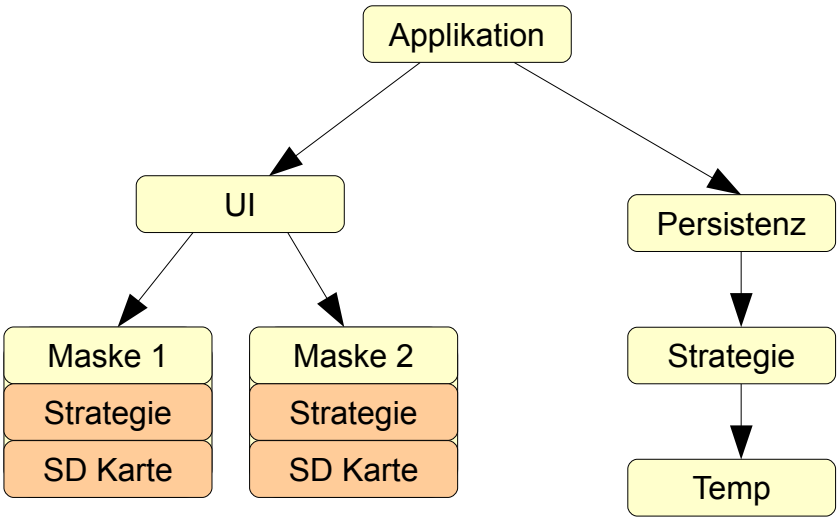

Abbildung 4: Die Redundanz wurde "entfernt"

Besser

b) Die Speicherung der über die <Maske> eingegebenen Daten wird von dem Persistenzsubmodul übernommen. .....

(2 Seiten weiter)

Die Applikation soll selbständig eine eingesteckte SD Karte erkennen, in diesem Fall sind alle Applikationsdaten automatisch auf der SD Karte zu speichern. Sollten bereits temporäre Arbeitsdaten vorhanden sein, so sind diese auf die Speicherkarte zu übertragen.

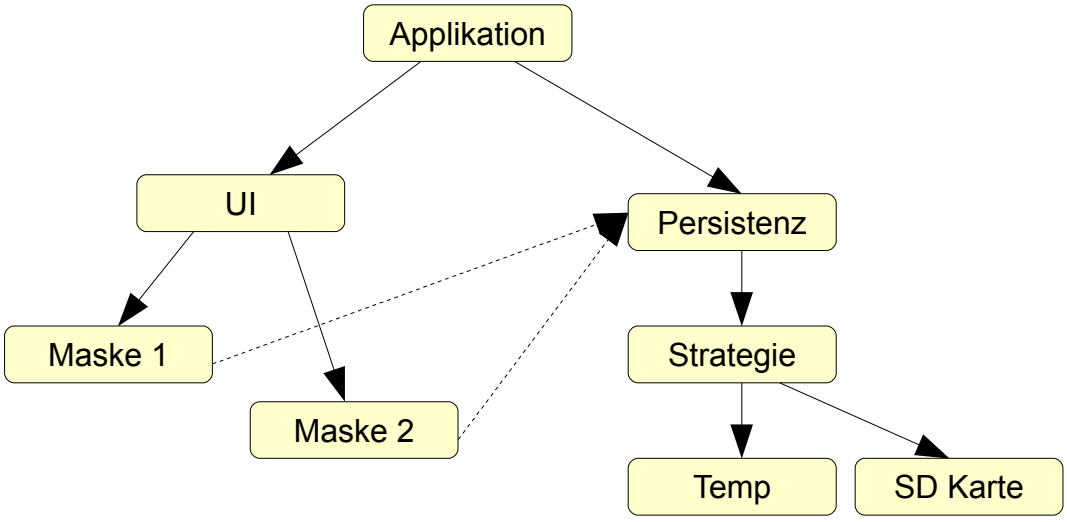

Abbildung 5: Korrekt strukturierte Analyse

Warum ist es besser? Grundsätzlich sind die beiden Fälle von der Aussagekraft her gleich. Bei der näheren Betrachtung ergeben sich bei dem Fall b) mehrere Vorteile:

- Die Speicherstrategie muss nur einmal angegeben werden (Redundanz). Bei einer

Änderung der Strategie muss nur eine Stelle in der Analyse geändert werden (Abbildung 3)

- Der Leser wird beim Lesen nicht mit an dieser Stelle nicht notwendigen Details belastet, was dem Textverständnis zu gute kommt (Komplexität).
- Unter Umständen wird die Strategie nicht mehr bei der Analyse der Persistenz erwähnt (es Stand ja schon bei der UI). Was dazuführen kann das dies bei der Implementierung übersehen werden kann (Abbildung 4)
- Weiterhin kann es zu Widersprüchen kommen, falls Gegebenheiten mehrfach (unterschiedlich) beschrieben werden (Abbildung 3).

1.2.3 Projektbeschreibung

Die kurze Projektbeschreibung sollte dem Leser einen kurzen Überblick über das der Analyse zugrunde liegende Problem geben. Bei kleinen Projekten kann es sich einfach um den Projektnamen selbst handeln (a). Falls die Zuordnung nicht eindeutig ist (mehrere ähnliche Projekte), bzw. die Vermutung besteht, dass der Leser die Analyse einem Projekt nicht eindeutig Zuordnen kann (z.b. 20-te Erweiterung), ist dies mit weiteren Informationen anzureichern (b).

a) *Bei der Analyse des Projekts <Projektname> wurde folgendes festgestellt.....*

b) Im Rahmen des Projekts <Projektname> für die <Auftraggeber> soll die bereits vorhandene Anwendung um <weitere Identifikationsmerkmale z.b. Versionsnummer > erweitert werden...

1.2.4 Abgrenzung

Die Abgrenzung in der Analyse verfolgt zwei eng miteinander verbundene Ziele. Beide Ziele beschreiben den Umfang der Arbeiten, die im Rahmen des Projekts **nicht** gemacht werden müssen, weil sie entweder bereits vorhanden sind bzw. extern eingekauft werden können (a), oder von anderen Entwicklergruppen bearbeitet werden (b). Dabei ist es wichtig zu erwähnen, wer für diese Arbeiten zuständig ist (Zulieferer, andere Abteilung) so dass bei Bedarf die notwendigen Quellen zur Informationen bezüglich dieser Projektteile bekannt sind.

Die Abgrenzung zu anderen Projektteilnehmern sollte des Überblicks wegen möglichst früh in der Analyse vorgenommen werden. Die Abgrenzung der verwendeten Fremdentwicklungen (Bibliotheken, Frameworks) eher bei Bedarf, ausgenommen die Fälle, in welchen das Projekt auf dieser Fremdentwicklung basiert.

a) Da die im Hause vorhandenen Entwicklungskapazitäten für die Implementierung ... nicht in ausreichendem Umfang vorhanden sind, soll die <Funktionalität> extern als fertige Bibliothek eingekauft und ins Projekt eingebunden werden.

b) Hierzu soll von der Entwicklungsabteilung nur das Grundgerüst, bestehend aus der Hauptanwendung und den frei konfigurierbaren Eingabemasken, implementiert werden. Die kundenspezifische Konfiguration der Masken selbst soll später, den Kundenwünschen entsprechend, von der Projektierung projektbezogen durchgeführt werden.

1.2.5 Vorgaben

Um den Kontext der Analyse korrekt auffassen zu können sind die Vorgaben des Auftraggebers zu beachten und entsprechend auch zu notieren. Diese Vorgaben können nicht nur die Software selbst, sondern acht den Entwicklungsprozess an sich bzw. dessen Teile (b) betreffen. Diese sind bei der Analyse auch entsprechend einzuhalten. Des weiteren soll dies beim Entwurf sowie der Implementierung helfen nicht vorgabenkonforme Lösungswege von vorne auszuschließen (c).

a) Damit die Motorsteuerung die interne Qualitätsprüfung des Auftraggebers bestehen kann, ist diese in C99 zu erstellen. Des weiteren ist die vollständige Einhaltung der MISRA-C Spezifikation vorgegeben.

b) *...Der Entwurf des Moduls <BrauchenWir> ist im UML Diagramm nach ISO/IEC 19505 zu dokumentieren.....*

c)*...Da die Ausrüstung unter anderem für die Arktisexpedition vorgesehen ist, können nur „military grade“ ICs verwendet werden, da nur für diese die störungsfreie Funktion in dem benötigten Temperaturbereich (-55° bis 125°) seitens des Herstellers garantiert wird.*

1.2.6 Gesetze und Normen

Es ist wichtig bereits bei der Analyse zu evaluieren ob die zu entwickelnde Software oder Teile davon gesetzlichen Regelungen bzw. sonstigen Vorschriften oder Normen (DIN / ISO) unterliegt. Sollte der Fall zutreffen ist dies in der Analyse entsprechend hervorzuheben.

1.2.7 Rückfragen

Rückfragen an den Kunden und die daraus entstandenen Antworten sollten in der Analyse enthalten sein, vor allem wen sie zum Verständnis der Aufgabenstellung notwendig sind. Sie dienen auch zur Begründung warum bestimmte Varianten gewählt bzw. verworfen wurden. Dabei sollte die Frage sowie die dazugehörende Antwort nach Möglichkeit zusammen gehalten werden.

Von der Aufgabe zur Anwendung

....Bei der genaueren Betrachtung des Sachverhalts stellte es sich heraus, dass ... realistisch nicht mit vertretbaren Aufwand realisierbar ist, weil.... Aus diesem Grund, soll, nach Rücksprache mit dem Kunden ..... umgesetzt werden.

Auch hier gilt es zu unterscheiden ob es sich um eine Frage von grundsätzlicher Tragweite handelt. Diese wird dann entsprechend im Kopfteil der Analyse enthalten sein. Detailfragen werden an der Stelle der Analyse behandelt, an der das Umfeld dieser Frage untersucht wird.

1.2.8 Algorithmen

Die Algorithmen sind in der Analyse zu behandeln falls es darum geht, die genaue Funktion eines bestehenden, vorgegebenen Verfahrens zu ergründen (Analyse des Algorithmus selbst). Des weiteren können in der Analyse auch die zu Verfügung stehenden Algorithmen aufgelistet und auf ihre Verwendbarkeit untersucht werden.

Um die geforderte Genauigkeit der Berechnung einhalten zu können sind Fließkommazahlen ungeeignet, da sie nur eine Genauigkeit von 53 Bit ermöglichen. Statt dessen soll auf Festkommazahlen zurückgegriffen werden. ..... Dabei ist bei der Multiplikation zu beachten, dass das Ergebnis einer 64 Bit Multiplikation 128 Bit lang ist, so dass es um die Nachkommastellen mit einem Bitshift nach Rechts korrigiert werden muss.....

Auch etwaige Einschränkungen der Algorithmen, falls diese bereits bekannt sind, sollten hier nicht verschwiegen werden.

... bei der Durchführung von Testberechnungen wurde festgestellt, dass der Algorithmus für x>3000 nicht mehr die gewünschte Genauigkeit bietet. Da systembedingt keine x Werte die höher als 2500 vorkommen sollen, ist dieser Algorithmus zur Lösung der Aufgabe dennoch geeignet. Es sollte jedoch sichergestellt werden das seitens des Benutzers keine Eingabewerte höher als 3000 eingegeben werden können.

Ein noch zu entwickelter Algorithmus wird in der Entwurfsphase beschrieben. Ausgenommen davon sind einerseits die Tatsache dass dieser entwickelt werden muss, sowie die Eigenschaften die der zu entwickelnde Algorithmus besitzen soll. Diese Eigenschaften sind in der Regel zu begründen wie. z.b. Handelsgesetzbuch, DIN-Normen, Kundenvorgaben etc. .

.. Die Berechnung der <BWL lässt grüßen> ist Vorschriftsgemäß <HGB §xyz> intern mit vier Nachkommastellen durchzuführen, das Ergebnis der Berechnung ist auf zwei Nachkommastellen mit kaufmännischer Rundung zu kürzen.

1.2.9 Vorgänge / Verfahren

Da (Arbeits-)Vorgänge bzw. technische Verfahren spätestens nach der Implementierung zu Algorithmen werden, sind diese ähnlich zu behandeln. Ganz wichtig ist dabei die genaue Beschreibung von Vorgängen und Verfahren, die von der Software abgebildet werden sollen.

Folgender manueller Vorgang soll automatisiert werden:

Nach dem Drücken der „Starttaste" fährt der Wagenschlitten heraus. Ist der Schlitten voll ausgefahren, was durch den Schalter AJK-34-56 /2 festgestellt werden kann, senkt der Heber das Werkstück auf den Schlitten herab. Die auf dem Schlitten montierte Waage registriert das Gewicht des Werkstücks und ordnet es der entsprechenden Gewichtsklasse A,B oder C zu. In Abhängigkeit ..... . (es existiert, also „ist" Form).

...Dieses Verfahren soll folgendermaßen automatisiert werden: durch Einbau von zwei Sensoren soll die Position des Hebers, sowie das Vorhandensein eines Werkstücks im Heber automatisch erkannt werden. Ist Der Heber ausgefahren und mit einem Werkstück bestückt so soll der Schlitten automatisch ausgefahren werden. Nach dem erreichen der Schlittenendposition ist das Werkstück auf dem Wagen abzulegen..... (Dieses Verfahren ist neu, also „es soll")

1.2.10 Einschränkungen

Sollte bereits bei der Analyse auffallen, dass bestimmte Komponenten nicht miteinander kompatibel sind, oder in Ihrer Arbeitsweise eingeschränkt sind, ist dies entsprechend zu verzeichnen. Die Gründe dafür sind entsprechen anzugeben. Es ist auch wichtig den Zeitpunkt (oder die Version der Komponente) der Feststellung anzugeben, damit bei Erscheinen einer neueren Version diese evaluiert werden kann.

Nach ausgiebiger Internetrecherche wurde festgestellt, dass OpenGl und JavaFX im Moment (Stand Juni 2013) nicht miteinander Kompatibel sind. Aus diesem Grund kann dieses Projekt nur mit JavaSwing umgesetzt werden.

Diese Einschränkungen sind nach Möglichkeit zu belegen bzw. zu beweisen.

Laut Auskunft des Oracle Supports (Anhang B-4) ist derzeit (Juni 2013) das Verwenden des OpenGl Frameworks innerhalb von JavaFX nicht möglich. Aus diesem Grund kann dieses Projekt nur mit JavaSwing umgesetzt werden.

Dies gilt natürlich auch für alle anderen Aspekte der Analyse.

Von der Aufgabe zur Anwendung

...aufgetreten. Bei der Untersuchung des Problems wurde festgestellt, dass in der Hauptgeschäftszeit (9:00-15:00) nicht mehr als 2000 Datensätze pro Minute mit der Hauptgeschäftsstelle ausgetauscht werden können....<Begründung falls die Ursache dafür bekannt ist>

1.2.11 Fehler

Auf keinen Fall sind Fehlannahmen oder Fehler, die in der Analyse gemacht wurden, später aus dieser zu entfernen. Damit soll verhindert werden, dass bei nochmaligen Überarbeitung des Projekts die Idee aufkommt, es diesmal „besser“ zu machen und ein bereits gemachter Fehler wieder auftritt (ist dem Verfasser übrigens auch schon mal passiert). Es sollte auf jedem Fall der Grund für die nicht Umsetzbarkeit angegeben werden, so dass, falls dieser in der Zukunft wegfallen sollte, eine neue Prüfung durchgeführt werden kann.

...leider hat es sich bei der Implementierung herausgestellt, das auf Grund <Begründung> dies auf diese Art und Weise nicht umsetzbar ist. Deswegen soll an dieser Stelle ein anderer Algorithmus verwendet werden.....

1.2.12 Form

Grundsätzlich ist die Analyse nicht aus der Sicht des Autors, sondern eines unabhängigen Beobachters zu beschreiben. Deswegen sollen in der Analyse auch keine „ich“ , „wir“ oder „unsere“ vorkommen.

Im Unterschied zu dem Entwurf (es wird), oder Implementierung (es ist) sind der Text der Analyse eher als eine Reihe von Vorschlägen zu formulieren, es „soll“ ja was gemacht werden. Ob das durchsetzbar ist zeigt sich erst in erst in den weiteren Phasen. Ausgenommen davon sind bereits geschaffene Tatsachen, auf die bei dem Entwurf und Implementierung die Entwickler keinen Einfluss (mehr) haben, wie zum Beispiel bereits vorhandene Hardware.

Verwendete Fremdhardware:

Das das von KaufMich GmbH hergestellte Board verfügt über 3 Taster „Start“, „Stopp“, „Pause“. Diese können zur Steuerung des..... verwendet werden.

Eigene Entwicklung:

Die Konsultation mit der Entwicklungsabteilung hat ergeben das der zu entwickelte Board über 3 Taster verfügen soll - „Start“, „Stopp“ sowie „Pause“, die zur Steuerung des.... benötigt werden.

2 Entwurf

Die Entwurfsphase bestimmt maßgeblich die Struktur der späteren Anwendung. Diese soll sicherstellen, dass die nach dem Entwurf erstellte Software später nicht nur den Anforderungen der Aufgabenstellung entspricht, sondern dass diese auch weiteren Kriterien wie Erweiterbarkeit, Austauschbarkeit sowie Wartbarkeit entspricht. Das ist insofern wichtig, da die meisten Software Lösungen in Laufe ihres Lebens eine Evolution durchführen. Die Gründe dazu können unterschiedlicher Natur sein dazu gehören zum Beispiel:

- Zusammenfassung der einzelnen Betriebsvorgänge, um somit besseren Datenaustausch zwischen den einzelnen Arbeitsstationen oder Standorten zu ermöglichen.
- Neue technische Entwicklungen wie Internet, SMS, Mobile Geräte, VR
- Wechsel Laufzeitumgebung z.B. von Windows zu Linux oder zwischen verschiedenen Betriebssystemversionen (win95 -> win10)
- Allgemeiner Wunsch zu besserem Usability
- Erweiterung der Funktionalität oder weitere Automatisierung bestimmter Vorgänge z.B. DB basiertes Autovervollständigen bei Eingaben.

Ein weiterer Grund für den strukturierten Entwurf ist, dass dieser die spätere Implementierung deutlich vereinfacht, so dass die beim Entwurf „verlorene“ Zeit, in der Regel schon bei der Implementierung wieder zurückgewonnen wird, indem die Implementierungsphase deutlich kürzer ausfällt. Anschließend ist noch zu sagen, dass eine schlecht programmierte Anwendung, der aber ein gut durchdachter Entwurf zu Grunde liegt, in der Regel einer gut umgesetzten aber schlecht strukturierten Software deutlich überlegen ist. Wieso? Bei einem guten Konzept können die fehlerhaften Teile sukzessiv durch neue Implementierungen ausgetauscht werden. Bei einem schlechten Entwurf, muss nicht nur der Entwurf selbst korrigiert werden, sondern auch die darauf basierten Implementierungen!

2.1 Strukturierung des Entwurfs

In der Abbildung 3 sehen wir eine leider nicht selten anzutreffende Struktur einer Anwendung. Hier ist wohl aus „Zeitgründen“ auf jegliche Strukturierung der Anwendung verzichtet worden. Stattdessen werden die für die Verarbeitung notwendigen Daten an allen möglichen Stellen in der Software aus der „Welt geholt“, sowie die Ausgaben an den Anwender „Entsorgt“ wie es dem Entwickler gerade mal passt. Da diese Software, wie es so üblich ist, mit der Zeit gewachsen ist stellt es jetzt ein (nett ausgedruckt) wildes Netzwerk aus Funktionsaufrufen dar.

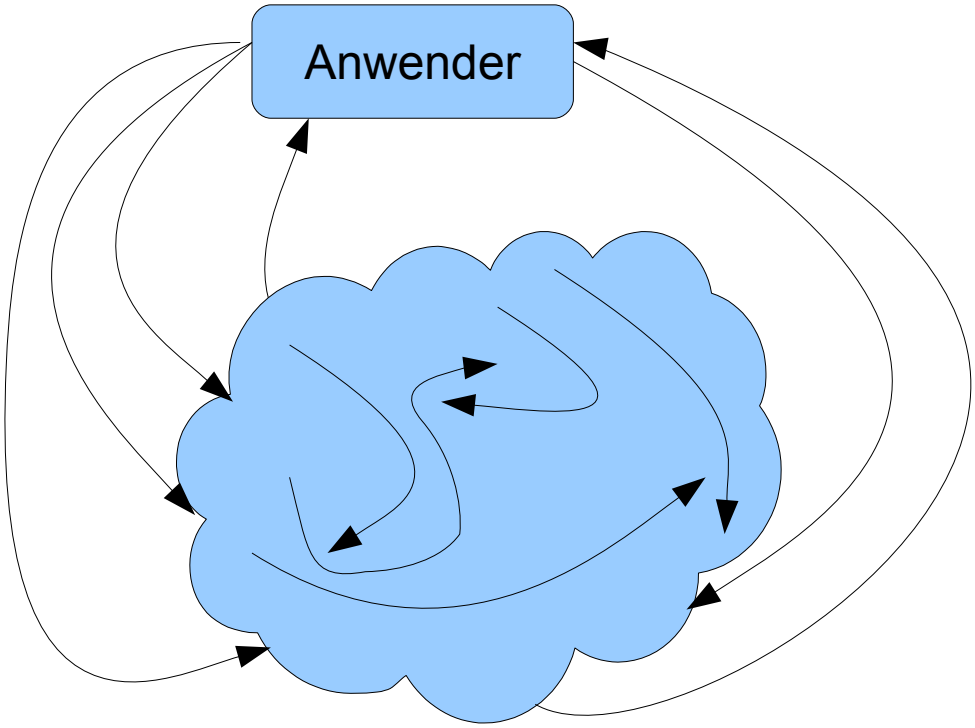

Abbildung 6: Strukturloser Entwurf

Da sich die einzelnen Programmteile untereinander auf unbekannte Weise beeinflussen, wird mit einer sehr hohen Wahrscheinlichkeit bei einem Versuch die Software zu erweitern (Abbildung 4) sich schon eine (meist mehrere) Funktion finden, der das nicht so passt. Mit etwas Glück bekommt es der Programmierer mit einer Warnung (kann ignoriert werden) oder einem Kompilierfehler (auf jedem Fall mit Arbeit verbunden) mitgeteilt. Jedoch in vielen Fällen bekommt man davon nichts mit, das System liefert allerdings auf einmal falsche Daten. Um solche Fehler herauszufinden, muss deswegen monolithisch konzipierte Software nach jeder Änderung komplett auf Seiteneffekte getestet werden. Nicht selten wir es sich beim Versuch die Software den neuen Gegebenheiten anzupassen herausstellen, dass eine an sich recht kleine Änderung massive Änderungen am System selbst verursacht. Das kann sogar so weit gehen, dass eine Neuentwicklung günstiger als eine Erweiterung des vorhandenen Systems bewertet wird. Beides ist aus betriebswirtschaftlichem Aspekt gesehen für ein Unternehmen ein Desaster und führte schon öfters zu einer Abwicklung des Unternehmens da es nicht mehr konkurrenzfähige Produkte wegen der Aktualität oder des Preises anbieten konnte.

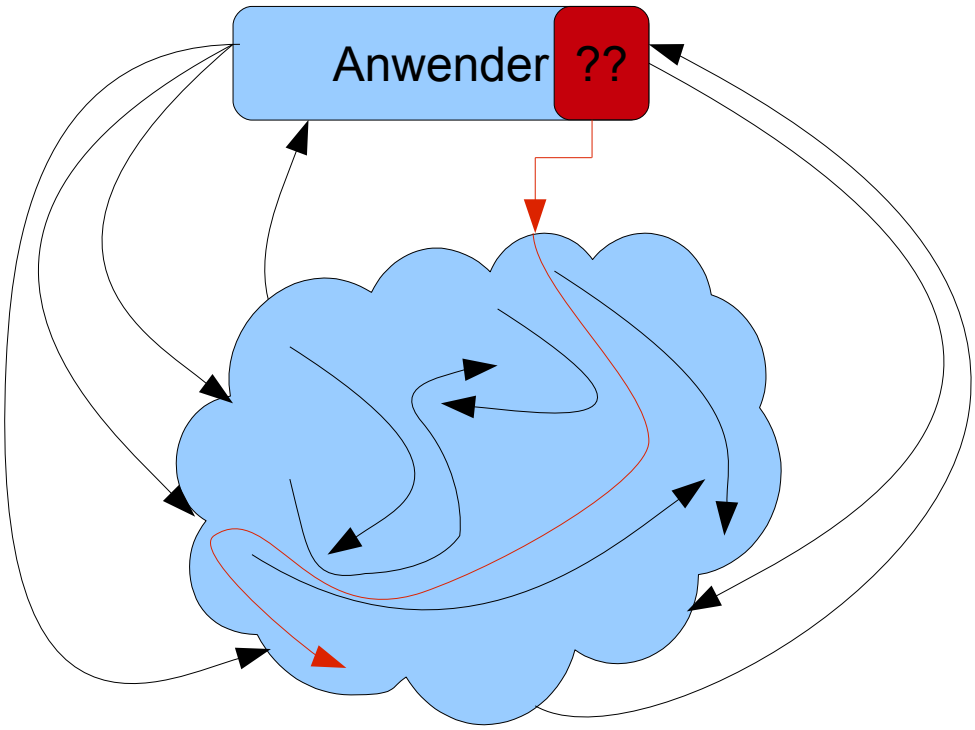

Abbildung 7: Strukturloser Entwurf wird erweitert

2.2 Vorteile einer Strukturierung des Entwurfs

Betrachten wir jetzt einen anderen Fall (Abbildung 5), in dem für den Entwurf zuerst (aus dem Blickwinkel des einen oder anderen vollkommen überflüssig) eine Menge von Ressourcen, hauptsächlich in Form von Arbeitszeit aufgewendet wurde. Wie wir sehen, hier wurden die Daten der drei Kunden zuerst „Normalisiert“ spricht in ein einheitliches Internes Format gebracht, danach sind die einzelnen „Geschäftsfelder“ der Anwendung abgegrenzt worden. Für die Kommunikation der einzelnen Elemente der Anwendung sind anschließend Schnittstellen, in der Zeichnung als Pfeile dargestellt, definiert worden.

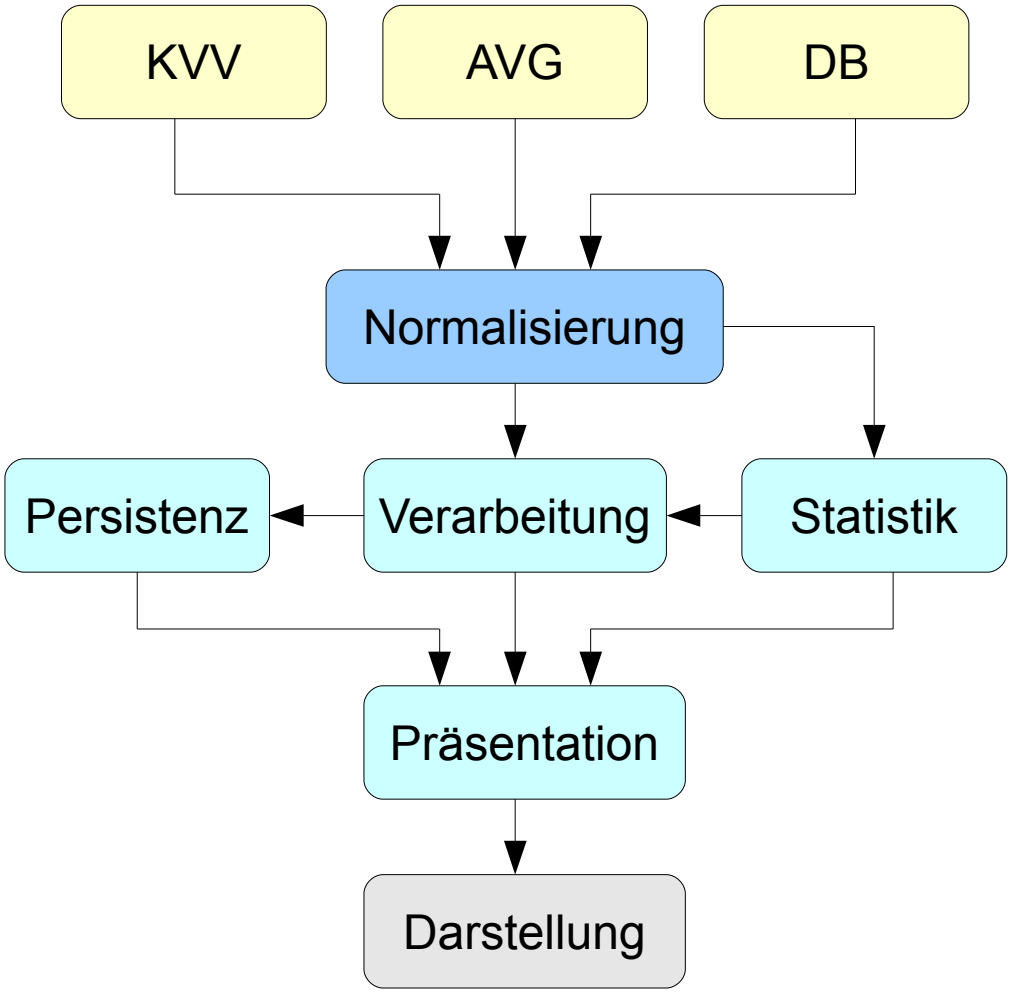

Abbildung 8: Eine gut strukturierte Anwendung

Eine derartige Konstruktion des Projekts bringt im Allgemeinen folgende Vorteile:

- Durch die klar definierten Schnittstellen können Programmteile problemlos von verschiedenen Entwickler (-gruppen) parallel entwickelt werden
- Die Komplexität der einzelnen Elemente ist um ein Vielfaches kleiner als die der Gesamtanwendung.
- Einzelne Funktionalitäten können physikalisch (Quelltext) besser lokalisiert werden.

- Fehlersuche wird vereinfacht, da die Fehlerquelle eher einer bestimmten Stelle zugeordnet werden kann.
- Änderungen an einem Element selbst beeinflussen die anderen Elemente nicht direkt.
- Die einzelnen Elemente können während der Entwicklungsphase auch einzeln getestet werden.
- Einzelne Elemente sind (z.B. kundenspezifisch) austauschbar z.B. für Kunden mit unterschiedlichen Datenbanksystemen.
- Es können problemlos weitere Funktionalitäten über neue Elemente hinzugefügt werden. Hierzu müssen „nur“ die entsprechenden Schnittstellen angepasst werden.
- Einzelne Programmierteile können in verschiedenen Programmiersprachen umgesetzt werden z.B. Verarbeitung in Java, Statistik in R, Präsentation in Qt die Verbindung der Elemente untereinander über Python.
- Es ist relativ schnell möglich Prototypen entweder als halb fertige Komponenten, oder unter Einsatz von Datengeneratoren den Kunden zu präsentieren. Die für das Produkt erstellten Elemente können in anderen Projekten direkt oder mit Anpassungen verwendet werden.

2.2.1 Prototyping und Tests

Gehen wir nochmals kurz auf die Problematik des Testens und des Prototyping. Als erstes schauen wir uns mal einen Prototyp an (Abbildung 6). Damit der Kunde ein Gefühl für seine spätere Anwendung bekommt, wurden die (teil-) implementierten Module „Präsentation“ und „Darstellung“ zur einer eigenständigen Applikation zusammensetzt. Somit kann der Kunde bereits mit der Anwendung „spielen“ um so festzustellen ob die Daten Ein- und Ausgabe seinen Erfordernissen entspricht. Dazu gehört zum Beispiel die Ergonomie wie z.B. Anordnung der Bedienelemente, Vollständigkeit der Ein und Ausgaben, die Vollständigkeit der in verschiedenen Ansichten (z.B.Fenster, Formulare) untergebrachten Funktionalitäten usw.

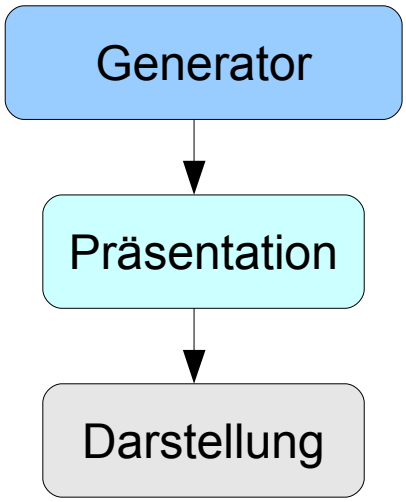

Abbildung 9: Prototyping

Die zweite Problematik (Abbildung 7), die kurz in diesem Zusammenhang an dieser Stelle behandelt werden sollte, ist der Aufbau der internen Tests. Wie sie in der Abbildung sehen können, wurden dort Kundendaten, die über einen bestimmten Zeitrahmen gesammelt wurden dazu verwendet, die gerade sich in der Entwicklung befindlichen Module zu testen. Des Weiteren könnte das System auch mit absichtlich falschen oder fehlerhaften Daten „gefüttert“ werden, um festzustellen, was dabei passiert und ob zum Beispiel die Anwendung bei fehlerhaften Eingaben immer noch stabil läuft.

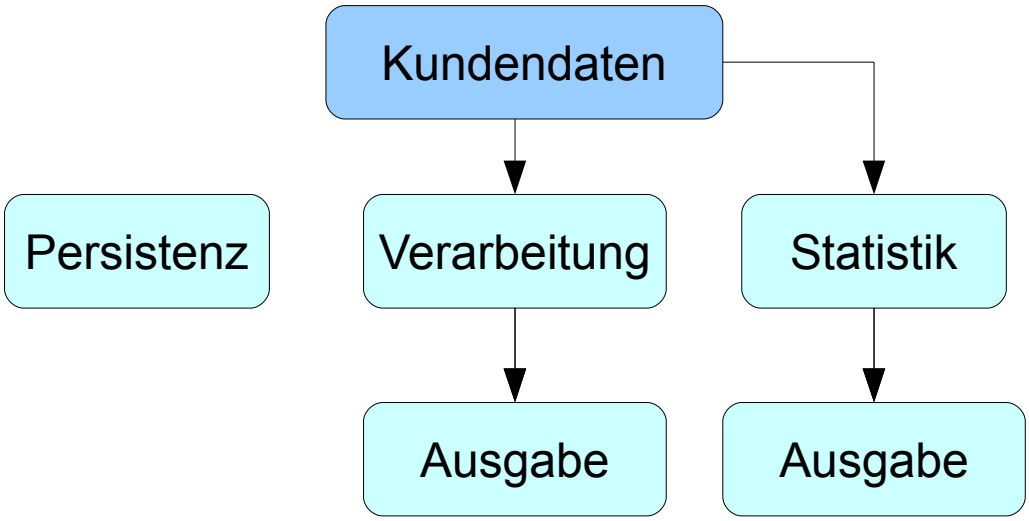

Abbildung 10: Testen mittels Kundendaten

Durch dieses Verfahren könnte die Vorgehensweise bei der Implementierung folgendermaßen aussehen:

1) Rumpf des Moduls erstellen
2) Testphase 1 - kommen die Daten am Modul an?
3) Modul an das Testsystem anschließen
4) Daten an **eine** Funktion, die diese verarbeitet, übergeben
5) Testphase 2 – Wird diese Funktion überhaupt aufgerufen?
6) Testphase 3 – Kommen die korrekten Daten bei dieser Funktion an?
7) Funktion schrittweise implementieren und überprüfen, ob die gerade implementierten Datenmanipulationen auch das gewünschte Ergebnis liefern.
8) Nächste Funktion implementieren

Diese Tests können auch direkt, falls diese Möglichkeit vorhanden ist, in der Entwicklungsumgebung („Debugger“) ablaufen. Dabei gibt es hauptsächlich zwei grundlegende Methoden. Einerseits können die Werte direkt in die Variablen zum Testen eingegeben werden. So einen Test sehen wir in der Abbildung 8, wo ein Test durchgeführt wird, bei dem der Laufzeitwert (88) größer ist als das eigentliche Maximum (7). Für alle die jetzt den Test für unsinnig halten – stellen wir uns mal vor ‚der Wert von *periodCounter* wäre 129, der *periodCounter* als 8 Bit Vorzeichenbehaftet und die Grenze als 128 definiert....

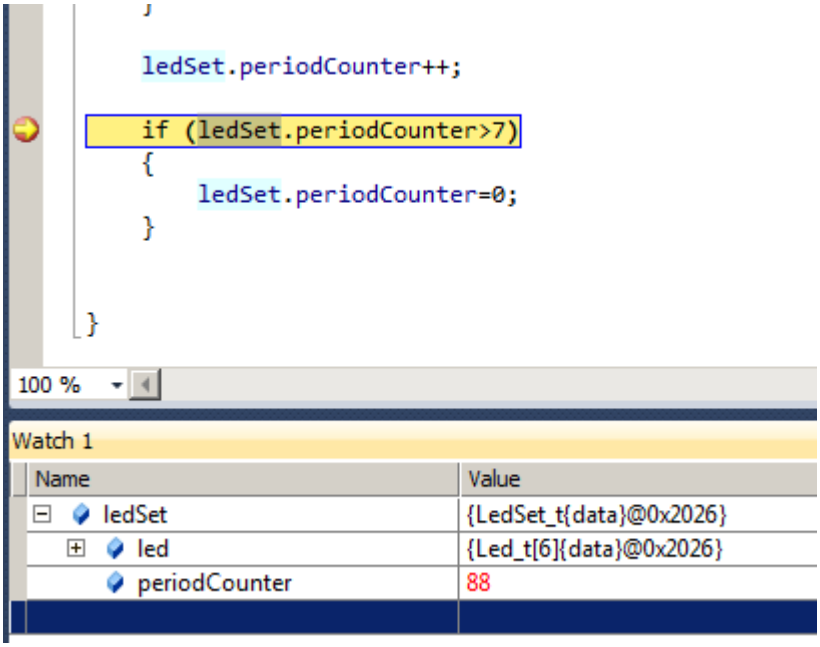

Abbildung 11: Datenmanipulation mit dem Debugger

Die andere Methode besteht darin, eine Testvariabel einzubauen und dabei bei C/C++ das Schlüsselwort ***volatile*** nicht vergessen, sonst optimiert es der gcc Compiler sehr gerne heraus. Diese Variable kann man dann wie in der Abbildung 8 zu sehen dazu verwenden um das Verhalten des Systems in bestimmten Situationen zu testen. In diesem Beispiel wird untersucht, ob bei bestimmten Obergrenzen Seiteneffekte vorkommen.

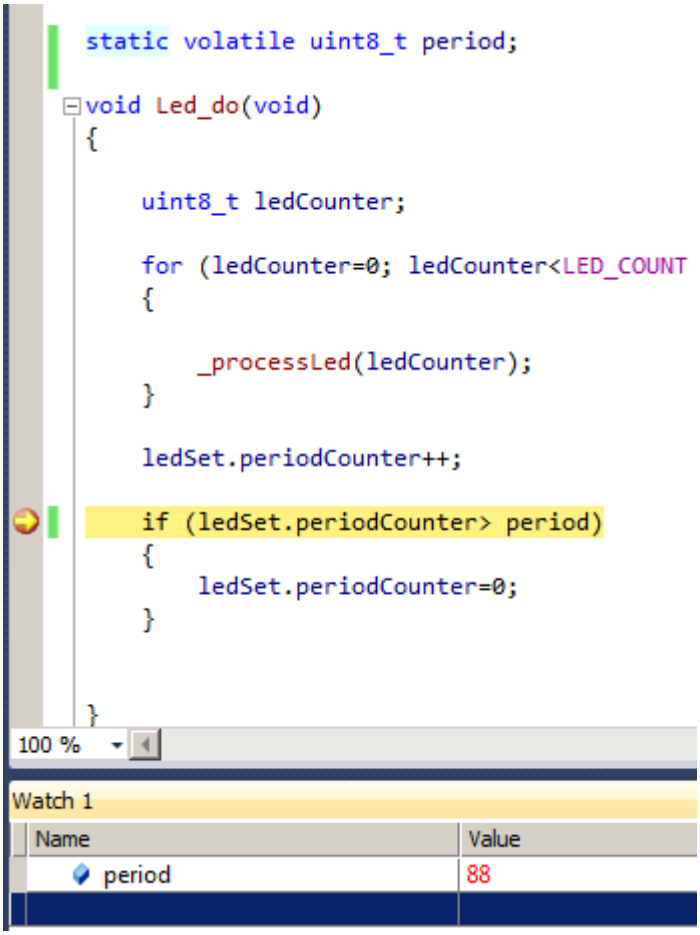

Abbildung 12 Testen mit Hilfsvariable

2.2.1.1 Top-Down

Als Top Down Methode wird ein Verfahren bezeichnet bei dem ein Problem durch Spezialisierung in kleinere einfachere Probleme zerlegt wird. Dies wird so lange sukzessiv wiederholt, bis die dabei entstandenen Probleme nicht mehr weiter (sinnvoll) unterteilbar sind. Da mit jeder Abstraktionstufe die Komplexität der Probleme sinkt, werden dadurch sogar sehr komplexe Vorgänge in der Implementierung beherrschbar. Betrachten wir das mal genau am Beispiel der Statistik (Abbildung 10) in unserer Beispielanwendung. Dabei kann die Statistik zuerst in drei Untermodule aufgeteilt werden:

- Fahrzeuge – alle Statistiken über die im Betrieb vorhandenen Fahrzeuge und dessen Zustand.
- Fahrgäste – wie viel Fahrgäste gibt es überhaupt, welche Fahrkarten werden wie oft verwendet (Einzelfahrkarte, Monatskarte, Umweltkarte etc.)
- Strecken – wie ausgelastet sind die einzelnen Strecken

Diese drei spezialisierten Statistik Untermodule sind noch ihrerseits in weitere Untermodule, die den einzelnen Rechenalgorithmen entsprechen, aufteilbar.

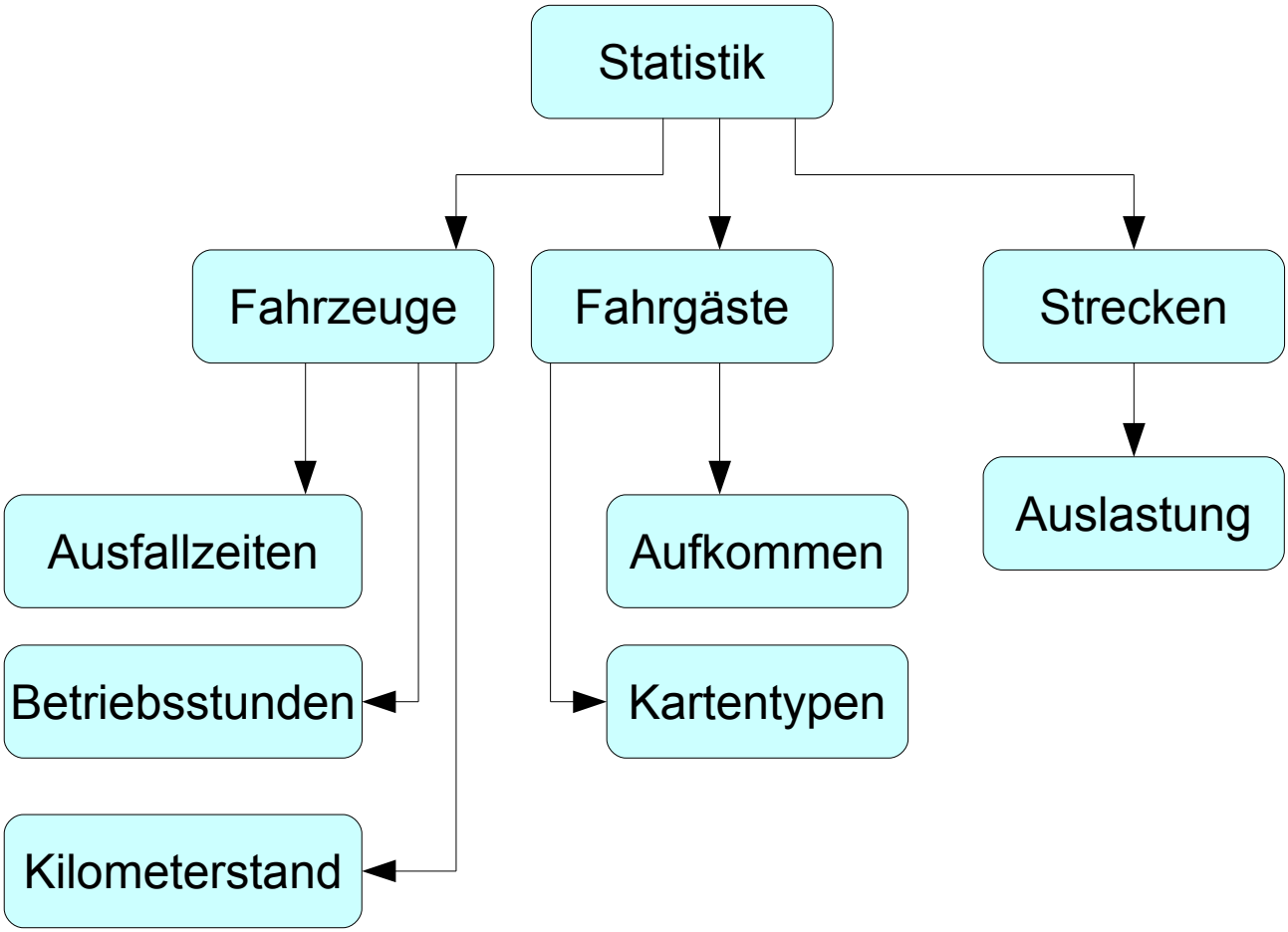

Abbildung 13: Top-down der Statistik

Auch die Rechenalgorithmen selbst lassen sich wiederum in weitere kleinere Einheiten aufteilen, wobei hier im Fall von Kilometerstandberechnung (Abbildung 11) nicht mehr eine Baumstruktur entsteht, sondern eine lineare Anweisungsfolge. Sollten alle Elemente dieser Folge (mehr oder weniger) Atomare Funktionen darstellen, kann hier mit der rekursiven Abarbeitung des Entwurfs abgebrochen werden. Dadurch ist es dem Programmierer jetzt direkt möglich die Implementierung der Funktion vorzunehmen. Wobei hier noch zusätzlich gleichzeitig der Prototyp der entsprechenden Funktion abgeleitet werden kann (Quelltext 1).

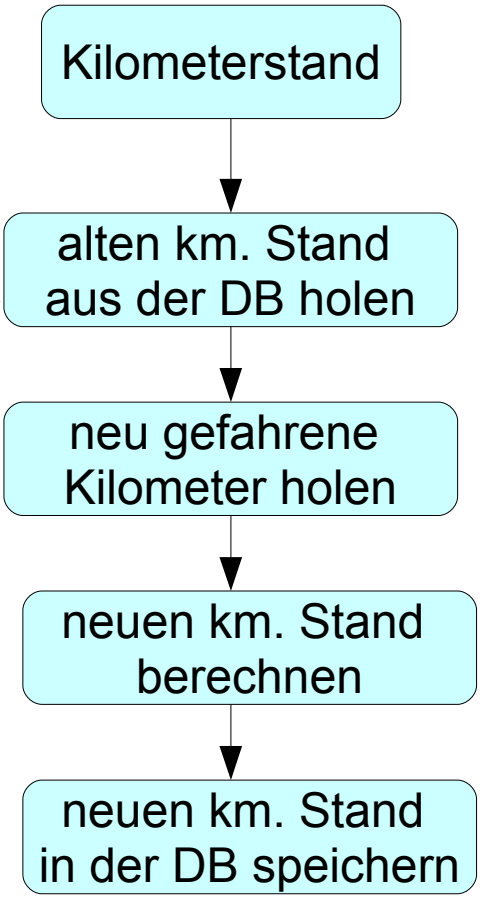

Abbildung 14: Top Down - Verfeinerung

```
KilometerstandStand::calculate()
{

        // Kilometerstand aus der DB holen
        // Dazugekommene Kilometer holen
        // Neuen stand berechnen
        // Neuen stand in DB zurückspeichern
}
```

Quelltext 1: Prototyp einer Funktion

In der Implementierungsphase werden dann die Kommentare durch entsprechende Anweisungen ergänzt (Quelltext 2). Wobei auch hier zum guten Programmierstil das sofortige Überprüfen, ob die richtigen Daten ankommen, gehört. Erst wenn dies gesichert ist wird der nächste Baustein implementiert.

```
KilometerstandStand::calculate(fahrzeugID_t id)
{
        VehicleData_t vehicleData;
        // Kilometerstand aus der DB holen
        vehicleData  = DB_getVehicleData(id);

        // Dazugekommene Kilometer holen
        // Neuen stand berechnen
        // Neuen stand in DB zurückspeichern
}
```

Quelltext 2

2.2.1.2 Priorisierung sowie Bottom-up

Wurde das zu implementierende Problem in einen Baum von Klein(-st)- Problemen aufgebrochen, sollten jetzt die Überlegungen zu der Implementierungsstrategie folgen. Dabei wird unter Berücksichtigung strategischer Gesichtspunkten die Reihenfolge der Umsetzung der einzelnen Unterprobleme bestimmt. Für gewöhnlich wird dabei in den einzelnen Ästen von unten nach oben vorgegangen und so diese nacheinander oder bei mehreren Entwicklern parallel implementiert. Die Folge in deren die Äste zu implementieren sind wird oft durch äußere Einflüsse oder Abhängigkeiten diktiert da z.B. ein Teilmodul von einer anderen Entwicklergruppe dringend für Ihre Arbeit (Test, Darstellung, Datenquelle) benötigt wird. Es sei an der Stelle angemerkt, dass dieser Vorgang vor allem durch die dabei notwendige Abschätzung der Implementierungsdauer der einzelnen Teile nicht trivial ist und oft nur mit viel Erfahrung korrekt zu lösen ist.

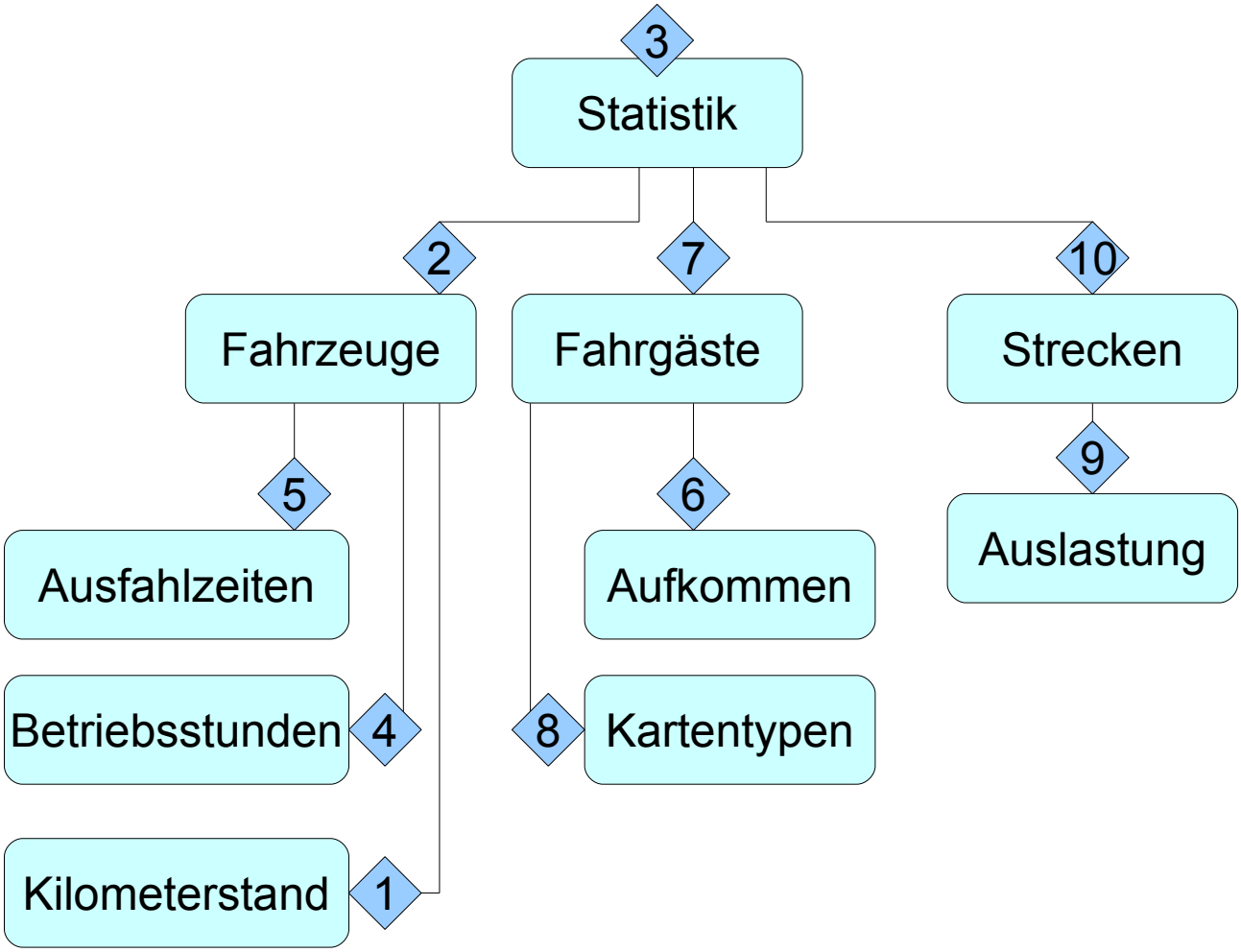

Abbildung 15: Priorisierung der Implementierung

2.2.1.3 Meilensteine

Aus der Priorisierung der Implementierung können direkt die Meilensteine abgeleitet werden. Dabei handelt es sich, wie der Name schon vermuten lässt, um Punkte an die bestimmte Entwicklungsphasen angeschlossen sind. Auch hier, genauso wie bei den anderen in diesem Dokument vorgestellten Techniken, gibt es verschiedene Abstraktionsstufen für die Meilensteine. Da gibt es normalerweise den „Masterplan“ der dem Kunden vorgestellt wird, und dieser dann das Erreichen bestimmter Meilensteine mit einer Überweisung honoriert. Des Weiteren, je nach Größe des Projekts, haben die einzelnen Arbeitsgruppen und Entwickler wiederum eigene Meilensteinlisten, die sie dann entsprechend abarbeiten. In der Tabelle 1 ist eine einfache Meilensteinliste abgebildet. In der linken Spalte sind die angenommenen bzw. geschätzten Zeiten zu denen die Meilensteine abgeschlossen werden sollten. In der mittleren Spalte ist der Umfang der Arbeiten, die zum Erreichen des Meilensteins notwendig sind, schematisch aufgezeichnet. Die rechte Spalte nimmt schließlich den aktuellen Status des Meilensteins auf. Wichtig ist bei etwaigen Problemen, die das Erreichen des Meilensteins verhindert haben, diese auch entsprechend schriftlich festzuhalten.

Datum	Aufgabe	Stand
10.1.2018	Kilometerstandstatistik funktionstüchtig	erledigt
15.1.2018	Fahrzeugstatistik implementiert	erledigt
20.1.2018	Statistikmodul implementiert	erledigt
21.1.2018	Betriebsstundenstatistik implementiert, Fahrzeugstatistik erweitert	erledigt
22.1.2018	Ausfallzeitenstatistik implementiert, Fahrzeugstatistik erweitert	Fehler (1)
24.1.2018	Fahrgastaufkommen implementiert	ja
1.2.2018	Fahrgästestatistik implementiert, Statistikmodul erweitert	in Arbeit
5.2.2018	Fahrkartenstatistik implementiert, Fahrgästestatistik erweitert	nein
8.2.2018	Streckenauslastungstatistik implementiert	nein
11.2.2018	Streckenstatistik implementiert, Statistikmodul erweitert	nein
20.2.2018	Test des Gesamtsystems	nein

Tabelle 1: Meilensteine der Statistik

Anmerkungen:

(1) konnte nicht getestet werden da Testdaten inkonsistent

In der letzten Zeile der Tabelle 1 ist die Aufgabe falsch formuliert es sollte eigentlich heißen „Gesamtsystem getestet“, da die Meilensteine den Ist- und nicht den Sollzustand definieren.

2.2.1.4 Erweiterung eines Entwurfs

Betrachten wir den Fall, dass einer der Kunden dieses System um ein Digitales Fahrauskunftssystem erweitert habe möchte, also diese sehr praktischen Teile, die so schön mit einer LED Fläche anzeigen, wann vielleicht die nächste Bahn theoretisch kommen könnte. (Abbildung 13).

Abbildung 16: Digitaler Fahrauskunftssystem

Abbildung 17: Fail - Ohne Lauftext !

In diesem Fall könnte es vielleicht so realisiert werden, dass über zusätzliche Daten die Information über den tatsächlichen Verbleib der Fahrzeuge in das System aufgenommen werden könnte (Abbildung 14). Danach in einem zweiten Schritt die Anwendung um die notwendige Vorhersage-funktion erweitert wird.

Dazu müssen, wie bereits festgestellt, zuerst die davon betroffenen Komponenten entsprechend erweitert werden. Dies kann in unserem Fall, anders als im vorhergehenden Monolithischen System (Abbildung 4), auch problemlos stufenweise erfolgen, indem einzelne Komponenten nacheinander erweitert und getestet werden können. Die von den Änderungen nicht betroffenen Komponenten müssen hingegen nicht mehr intensiv getestet werden. Es sollte nur ein anschließender kurzer Gesamttest durchgeführt werden. Damit kann man sicherstellen, dass der Anwender auch während der Entwicklungsphase der Erweiterungen gleichzeitig mit etwaigen erweiterten oder fehlerbereinigten Versionen des Programms versorgt wird.

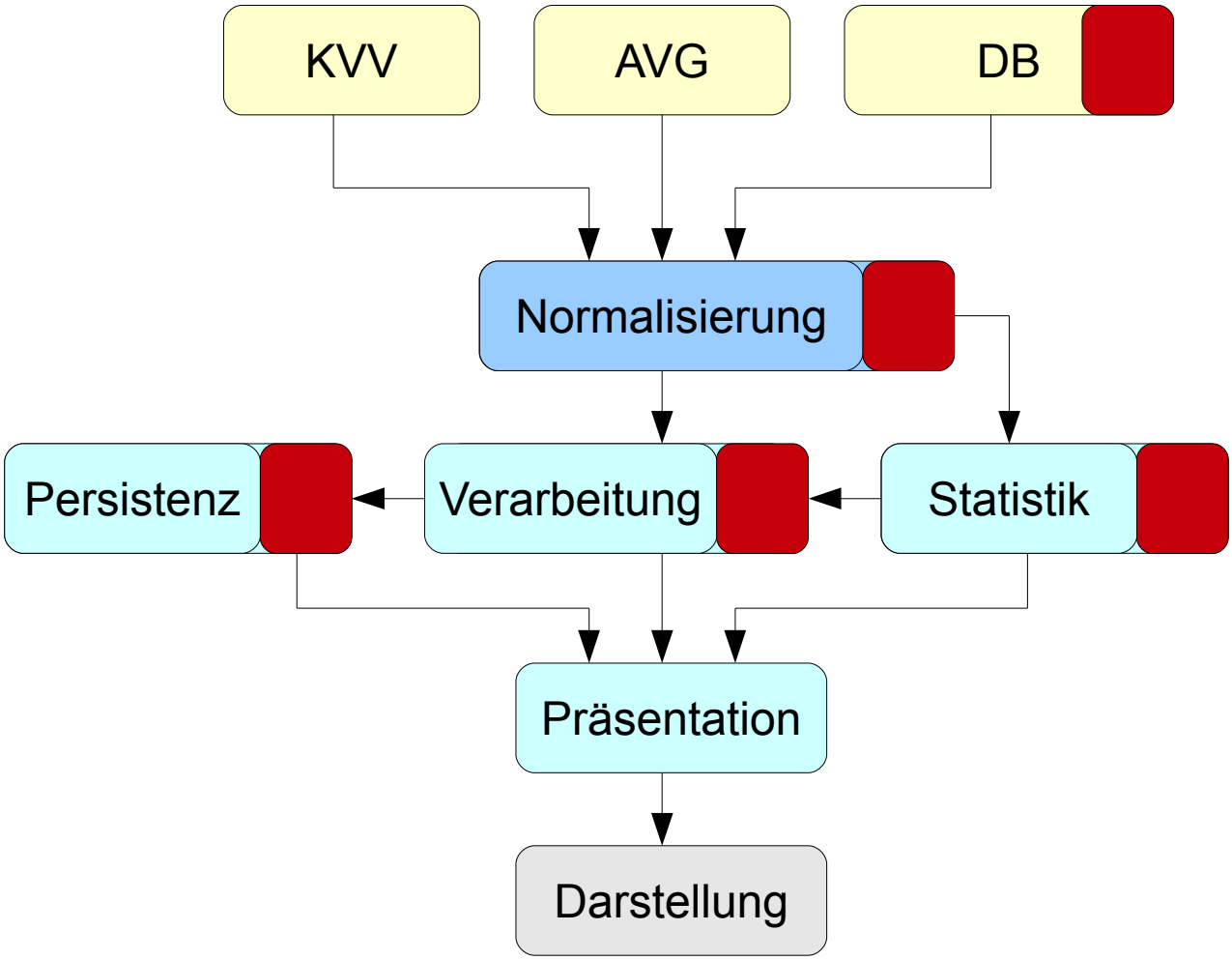

Abbildung 18: Erweiterter Strukturierter Entwurf (Daten)

In einem zweiten Schritt (Abbildung 15) werden zwei weitere Module der Anwendung hinzugefügt. Die Gewinnung der erwünschten Informationen aus den erweiterten Daten übernimmt das Vorhersage Modul. Hier können zuerst mal unterschiedliche Vorhersagemodule (Abbildung 16), die auf unterschiedlichen Ansetzen (Neuronale Netze, Genetische Algorithmen etc..) basieren implementiert werden. Durch die klar definierte Schnittstelle können dann alle diese Ansätze gleichzeitig ausprobiert werden und somit die Güte deren Vorhersagen bestimmt werden. Die Auswertung kann dann wiederum mit einem weiteren Modul automatisiert werden. Bei der Auswertung der Daten könnte natürlich auch vorkommen, dass in Abhängigkeit von äußeren Einflüssen wie Tageszeit, Baustellen, Wochentage, der Art der Vorhersage wie Kurzzeit, Langzeit bzw. Streckenlänge wie Stadtstrecke, Überlandstrecke die Güte der einzelnen Algorithmen unterschiedlich ausfällt ,so dass letztendlich sogar mehrere Algorithmen für unterschiedliche Vorhersagen nebeneinander in der fertigen Anwendung koexistieren.

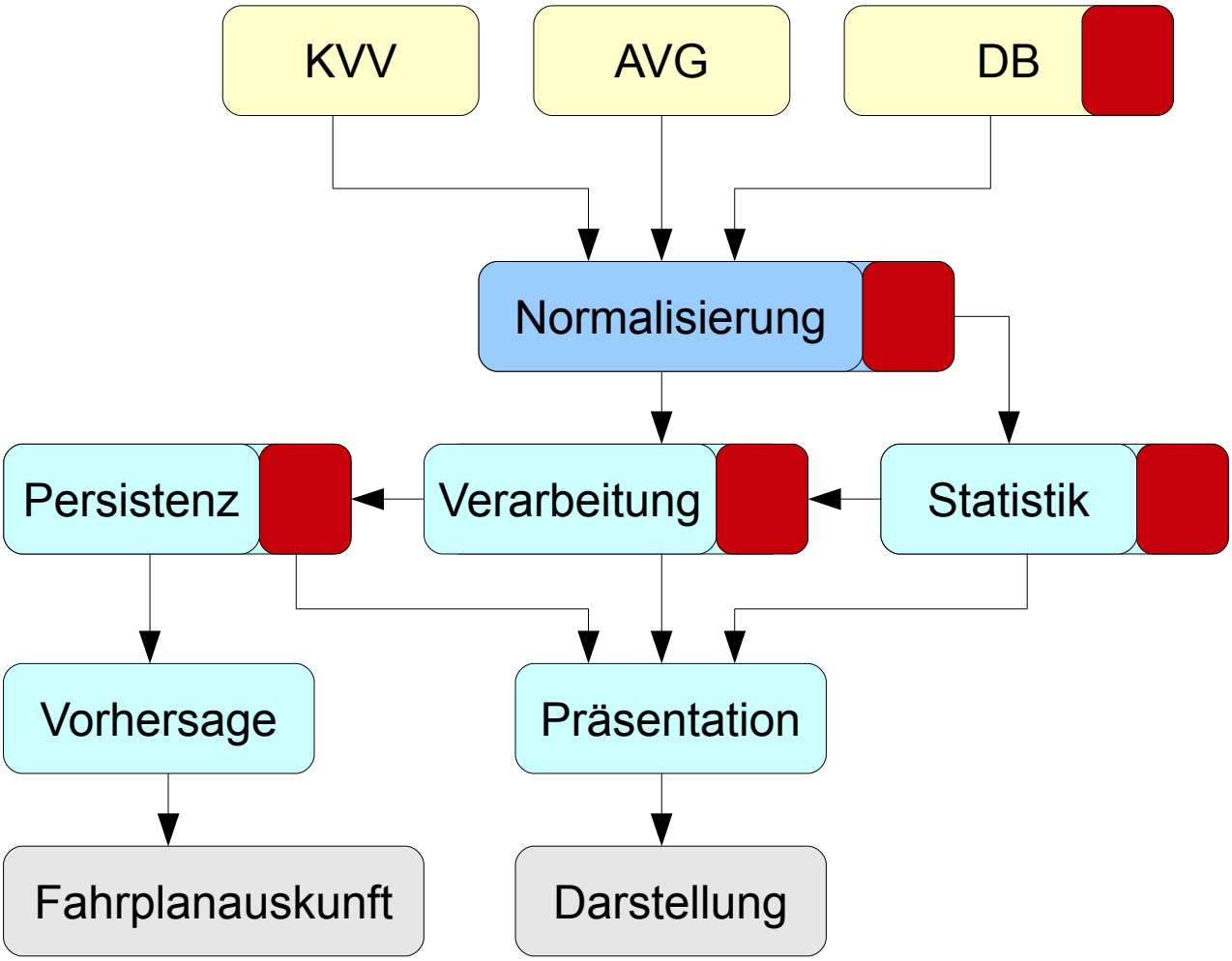

Abbildung 19: Erweiterter strukturierter Entwurf (Funktionalität)

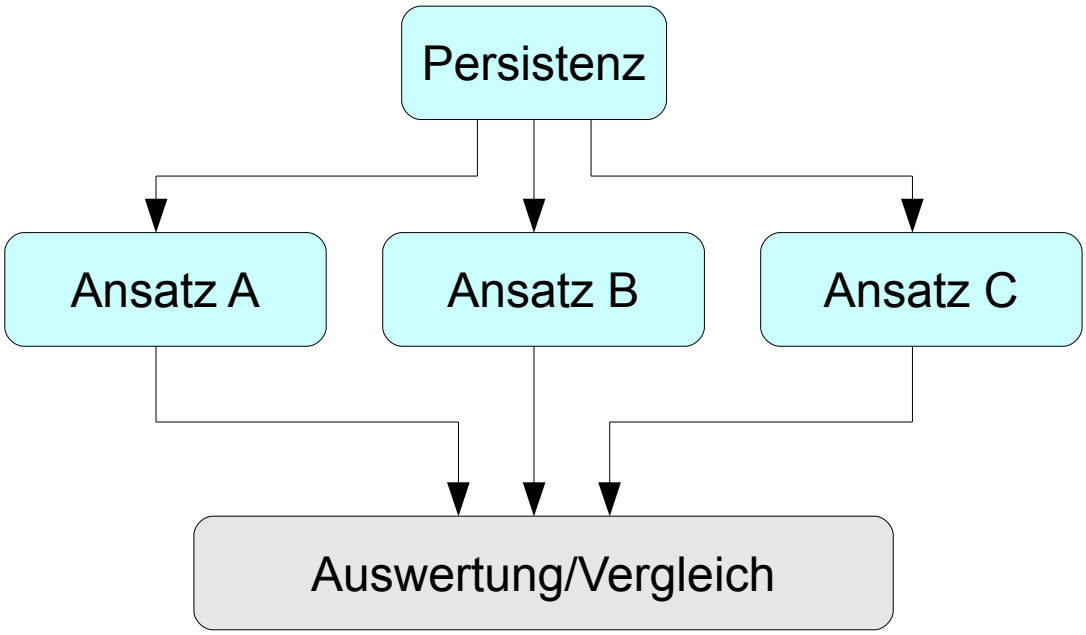

Abbildung 20: Testen unterschiedlicher Ansätze

2.2.1.5 Spezialisierung

Neben den bereits vorgetragenen Vorteilen ermöglicht ein gut durchdachter und strukturierter Entwurf eine nachträgliche Spezialisierung der Module. Stellen wir uns mal vor, dass der zuständige Sachbearbeiter auch beim Besuch in einem Reparaturwerk seine Statistikstellen nicht missen möchte. Dazu bieten sich in einer modernen Welt die sogenannten „Tablets“ an. So werden von dem Modul „Darstellung“ zwei weitere Implementierungen erstellt (Abbildung 17) - eine auf Android sowie eine auf iOS spezialisierte Version. Durch die bereits vorhandene Schnittstelle zwischen der Präsentation und der Darstellung können diese an dieser Stelle ins System „eingefügt“ werden. Je nach der Implementierung muss auch die Präsentation umgeschrieben werden, z.B. auf TCP/IP Kommunikation mit der Darstellung, aber dies beeinflusst den Rest der Anwendung in keiner Weise.

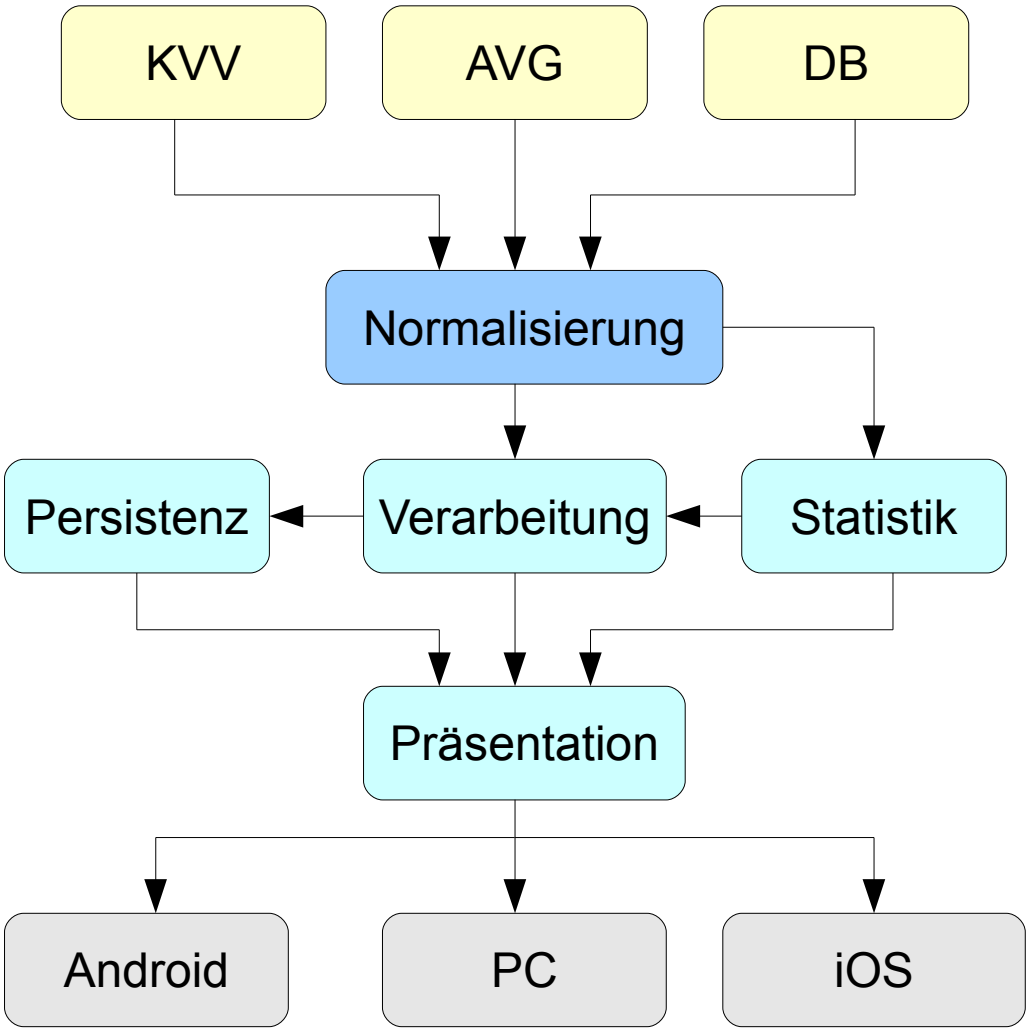

Abbildung 21: Spezialisierung der Module

2.2.1.6 Schnittstellen

Ein sehr wichtiger Teil des strukturierten Entwurfs stellen die Schnittstellen dar. Diese beschreiben die Kommunikation zwischen den einzelnen Elementen des Entwurfs. Sie stellen sicher, dass die Art, Umfang und Format der Daten, die zwischen den einzelnen Modulen ausgetauscht werden, von diesen auch korrekt interpretiert werden. Was die Schnittstelle selbst ist hängt hauptsächlich von der Kommunikationsart zwischen den Modulen ab. Im einfachsten Fall sind es einfache Funktionen; die zum Datenaustausch in den Modulen implementiert wurden (getter und setter). Es können aber auch Datenblöcke (Tabelle 2) sein, wenn die Module untereinander über Netzwerke kommunizieren (TCP/IP). Sehr wichtig ist auch, dass in der Schnittstelle nicht nur die Datentypen selbst, sondern auch noch die Einheiten der Daten definiert werden, damit diese auch korrekt interpretiert werden können.

Funktion : DB_fetchVehicleDataFromDB(vehicleID_t id)

Description : Get vehicleData_t structure with all statistic data for the vehicle with given id.

Parameter : id - id of vehicle

Return : vehicleData_t structure or null if no vehicle with given ID exsists

This function.....(hier folgt weitere genauere Beschreibung mit Einheiten, Wertebereichen Sonderfällen, Fehlerfällen usw.)

Abbildung 22: Beispiel einer Schnittstelle als Textbeschreibung

Index	Name	Typ	Wertebereich	Einheiten	Beschreibung
0	mesageID	UINT 16	0...65535		ID der Nachricht
2	messageType	UINT 8	0...100		Typ der Nachricht
3	vehicleID	UINT 16	0...10000		Fahrzeugkennung
5	actualVelocity	UINT 16	0...20000	((km/h) /100)-20	momentane Geschwindigkeit
7	...	...	...		

Tabelle 2 Beispiel einer Schnittstelle als Datenblock

3 Erstellung eines Entwurfs

3.1 Grobentwurf

Als erstes entsteht ein sogenannter Grobentwurf, in diesem werden die obersten Klassen der Probleme ausgemacht und entsprechend voneinander abgegrenzt. Es ist dabei zu beachten, dass diese als logisch zusammenhängende Einheiten zu betrachten sind. Beim Grobentwurf geht es eigentlich nur darum zu erkennen, welche Hauptaufgaben in der Gesamtaufgabe enthalten sind (Abbildung 23).

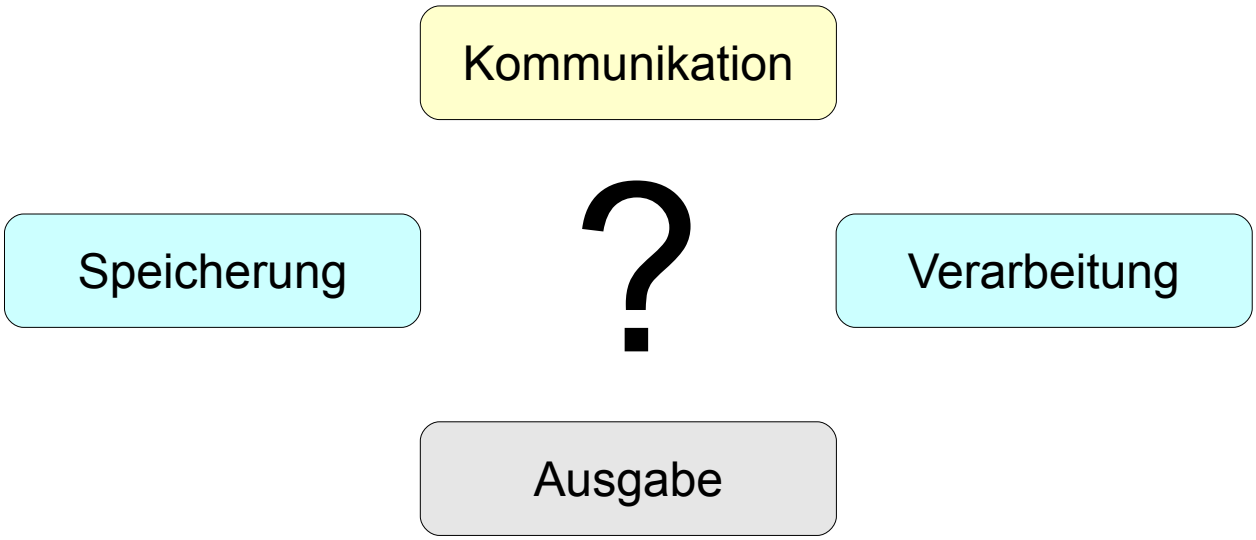

Abbildung 23: Grobentwurf

Als Beispiel für so eine zusammenhängende Einheit, soll hier das Kommunikationsmodul dienen. Die Kommunikation einer Anwendung ist für gewöhnlich gleichzeitig sowohl eine Dateneingabe, als auch Ausgabe. Nach dem strickten EVA Prinzip müsste diese streng getrennt voneinander implementiert werden. In der Praxis, wird dies allerdings meistens in einem Modul zusammengehalten. Der hauptsächliche Grund dafür ist, dass dadurch eine bessere Austauschbarkeit der Module erreicht werden kann, da so z.B. einfach die komplette USB Kommunikation einer Anwendung durch eine Bluetooth-basierte ersetzt werden kann, indem eine andere Kommunikation von Linker

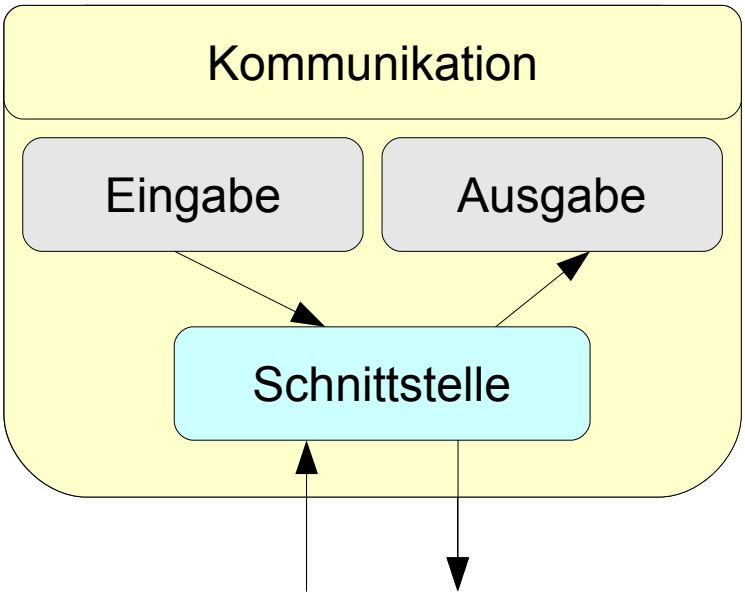

Abbildung 24: Kommunikationsmodul

eingebunden wird.

3.2 Feinentwurf

Sind die groben Eckdaten der Anwendung bekannt, dann kann nach der Top Down Methode mit der Verfeinerung der einzelnen Module begonnen werden. Bereits jetzt kann der Entwurf parallelisiert werden, indem die einzelnen Module an verschiedene Designer- oder Entwicklergruppen verteilt werden. Wichtig ist dabei, dass zuerst einerseits die Kompetenzen der Module, sowie anderseits die Schnittstellen, die ein Modul „nach außen“ anbietet, definiert sind. Jetzt kann damit begonnen werden nach der Top-Down Methode die einzelnen Module in Submodule, oder Funktionalitäten aufzubrechen (Abbildung 25). Dabei gibt es meistens mehrere gleichwertige Möglichkeiten wie ein Modul korrekt modelliert werden kann. Welche Modellierungsmöglichkeit letztendlich gewählt wird, hängt häufig von weiteren Faktoren ab. Einerseits könnte es sich dabei um persönliche Präferenzen des Designers handeln ein Problem auf eine bestimmte art und weise zu lösen, anderseits kann hier auch z.B. die spätere Erweiterbarkeit des Produktes eine Rolle spielen -dazu aber später mehr.

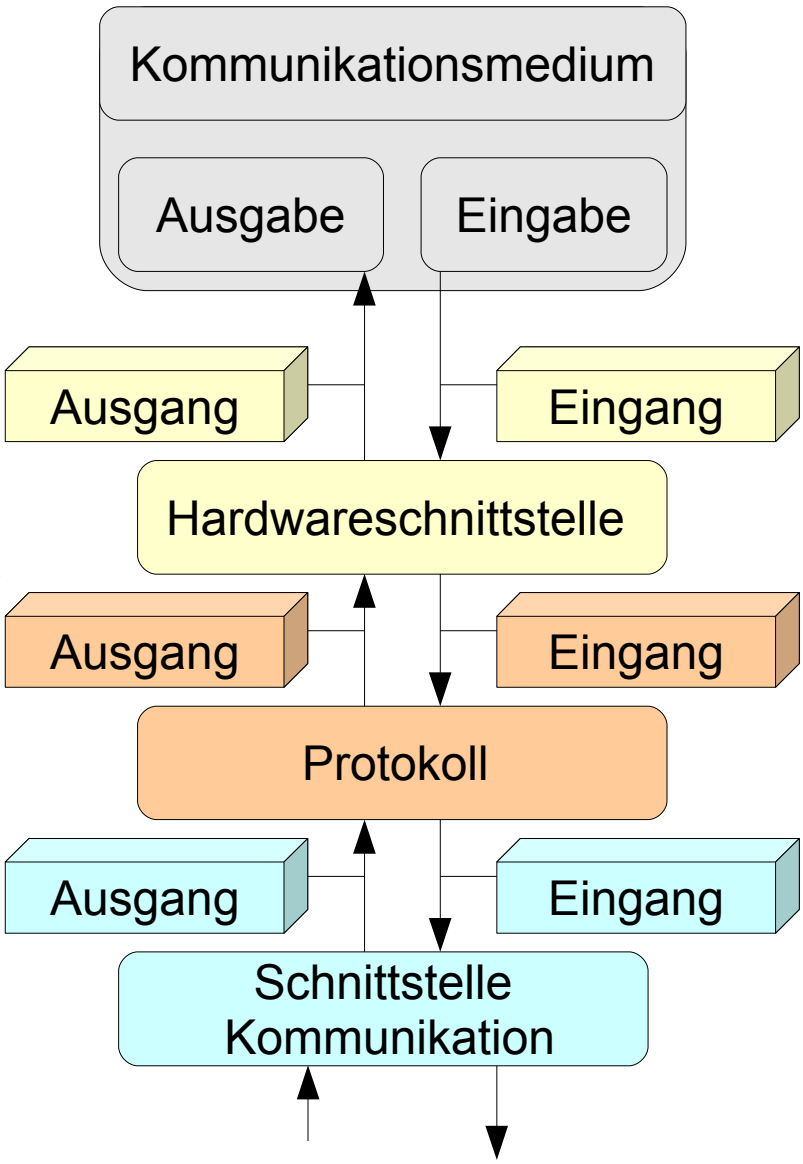

Abbildung 25: Feinentwurf des Kommunikationsmoduls

Wie in der Abbildung 25 zu sehen ist, durchlaufen die Daten der Anwendung einen mehrstufigen Abstraktionsprozess wobei dabei entsprechende Softwareschichten entstehen. Grundsätzlich ist es so, dass je mehr Schichten vorhanden , desto flexibler das Endprodukt. Die grundsätzliche Idee dabei ist, dass im Idealfall, die folgende oder vorausgegangene Schicht nichts von den „Problemen“ der aktuellen Schicht aufweist. In der Abbildung 26 ist hierzu als Beispiel der Verlauf eines Befehls der den linken Motor eines Roboters einschaltet.

Die Geschichte fängt damit an, dass eine höherliegende Anwendungsschicht den Motor einschalten möchte. Dazu wird die entsprechende Funktion in der Kommunikationsschnittstelle aufgerufen. Die Schnittstelle selbst, ruft die entsprechende Funktion im Protokoll auf. Diese generiert dann ein entsprechendes Datenpaket. Dabei entsteht das Paket [0x12,0x1] ([MOTOR_ON, LEFT_MOTOR].

Dieses Datenpaket wird weiter an die Hardwareschnittstelle gesendet, diese interessiert sich prinzipbedingt überhaupt nicht dafür, was die Daten, die übertragen werden müssen bedeuten. Diese Schnittstelle weiß aber dafür, dass damit die Gegenstelle den Befehl überhaupt als solches erkennen kann, zuerst ein Synchronisationsblock [0xFF,0xFF], sowie ihre eigene Adresse [0x03] übertragen werden muss. Sie generiert deswegen einen entsprechenden Header und hängt die zu übermittelnde Nachricht an den Header an. Danach übergibt diese das komplette Datenpaket an die nächste Schicht, die dann daraus den entsprechenden physikalischen Datenstrom aus elektrischen Spannungen erzeugt. Wie aus diesem Beispiel ersichtlich ist, können hier die Schichten nach Belieben ausgetauscht werden, z.B. das Protokoll wird getauscht, oder die Hardware Schnittstelle (z.b. von RS232 auf USB) wobei die anderen Module nicht geändert und neu getestet werden müssen. Soweit die Theorie – in der Praxis kommt es leider öfters auch vor, dass bestimmte Einzelheiten falsch modelliert, oder angenommen werden („Das braucht eh keiner“ so dass auch die anderen Schichten angepasst werden müssen, was aber auf jeden Fall deutlich einfacher ist, als bei monolithischer Software.

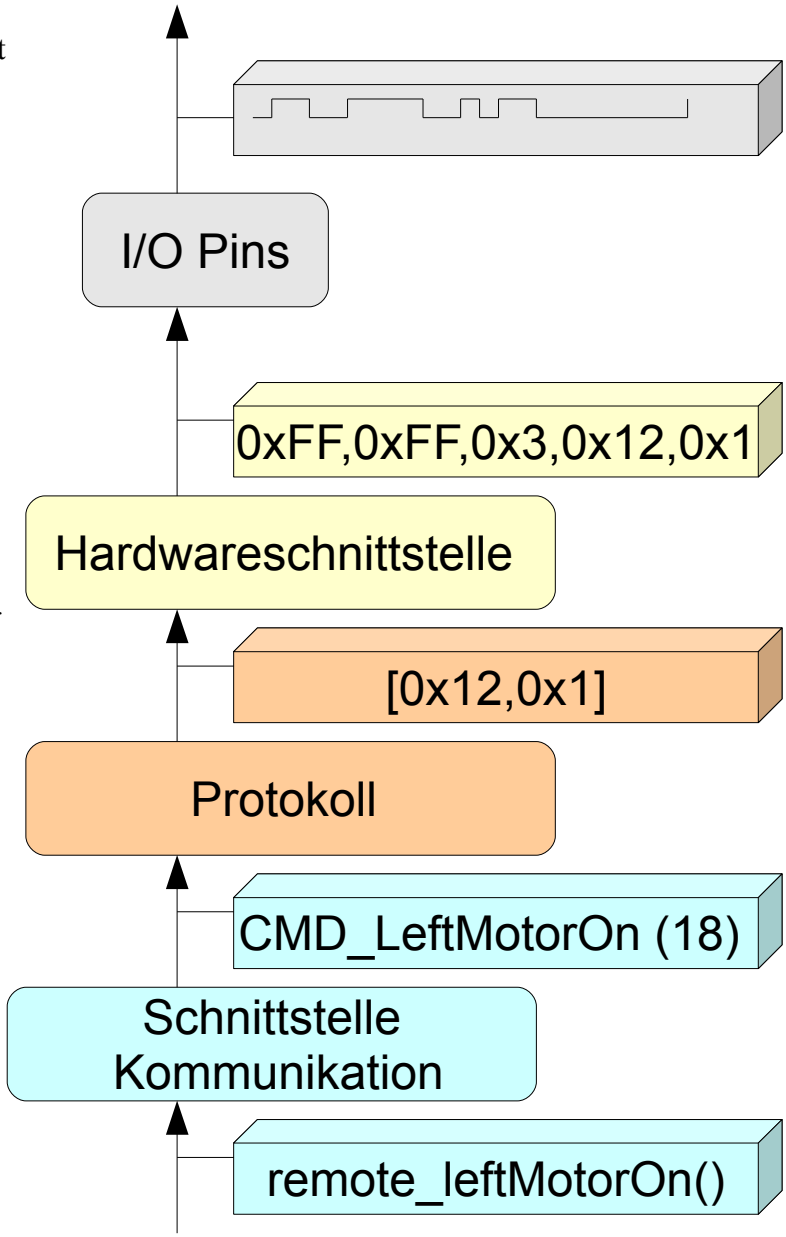

Abbildung 26: Abstraktionsschichten des Kommunikationsmoduls

Wie bereits erwähnt: Es gibt meistens mehrere Entwurfsmöglichkeiten. Das Beispiel in der Abbildung 25 wurde für ein System mit nur einem Protokoll entwickelt. Für ein System in dem z.B. mehrere Protokolle verwendet werden müssen, wäre das Beispiel aus der Abbildung 26 vermutlich besser, da der Austausch der Protokolle für verschiedene Kommunikationspartner deutlich einfacher realisiert werden kann. Aber auch das wäre mit dem ersten Entwurf möglich, allerdings würde

dadurch die Kommunikationsschnittelle selbst komplexer, da dort die Entscheidung über dem zu verwendeten Protokoll angesiedelt werden müsste.

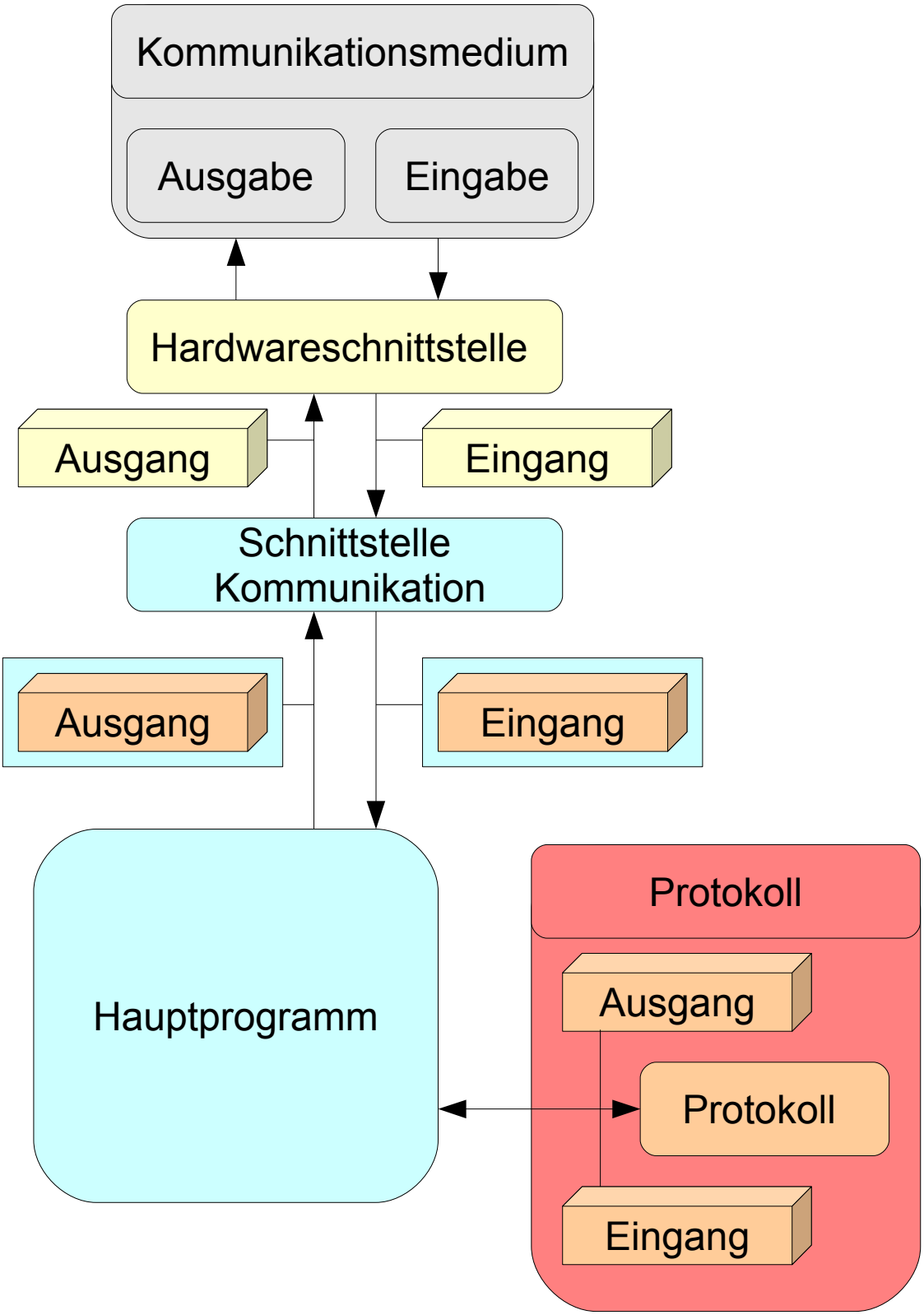

Abbildung 27: Alternativer Entwurf der Kommunikation

3.3 Spezialisierung

Sind die Hauptaufgaben der Module bekannt, kann im nächsten Schritt mit der Spezialisierung dieser begonnen werden. Es ist nicht selten so, dass für unterschiedliche Kunden, oder Einsatzgebiete angepasste Software erstellt werden soll. In diesem Fall gilt es die Module für die neuen Aufgaben anzupassen im einfachsten Fall wird ein vorhandenes Modul kopiert und danach entsprechend dem neuen Verwendungszweck umgeschrieben. Die Einbindung, der in der spezifischen Anwendung zu verwendeten Schnittstelle, kann entweder zu Kompilierzeit oder zu der Linkzeit erfolgen (Abbildung 28).

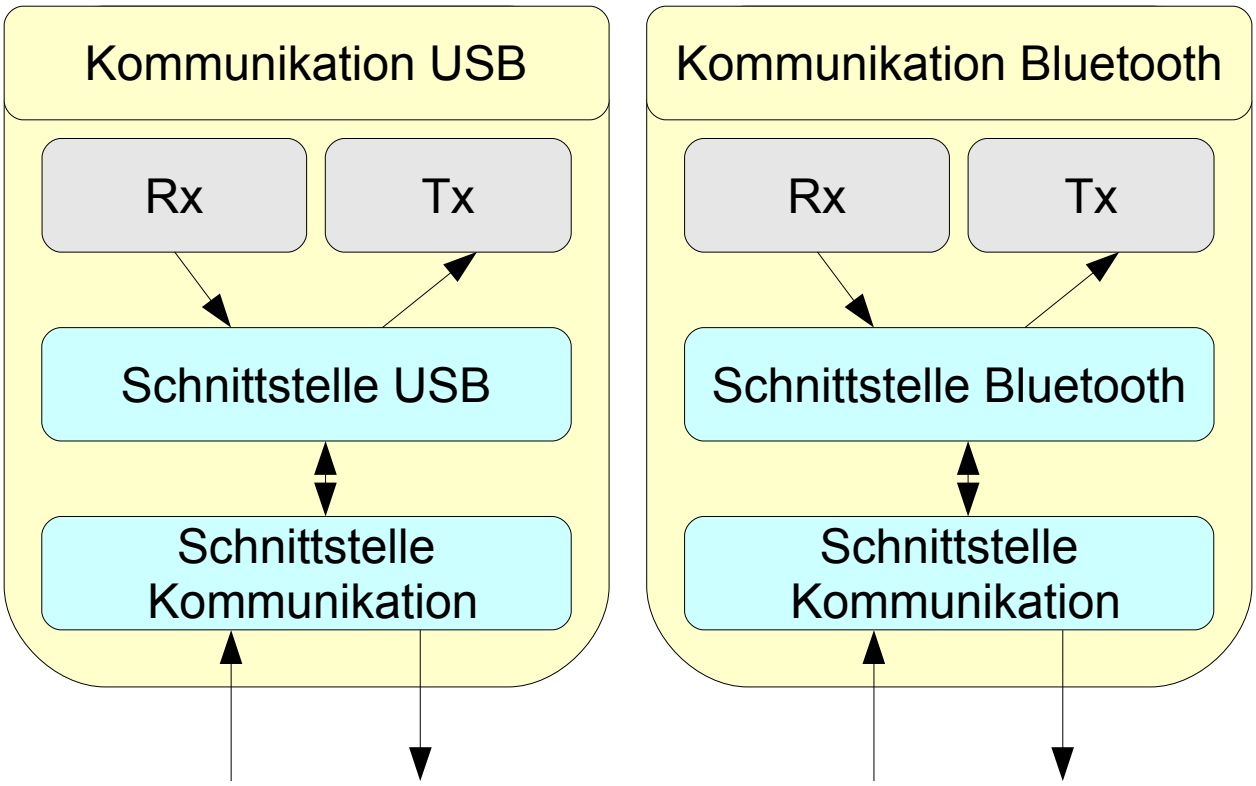

Abbildung 28: Spezialisierung der Kommunikationsschnittstelle

Dies lässt sich allerdings im zweitem Schritt noch verbessern, indem nur bestimmte Teile des Moduls um die neue Funktionalitäten erweitert werden, andere hingegen nur auf die Zusammenarbeit mit den neuen Funktionen „getrimmt“werden (Abbildung 29). Dies ist mehr oder weniger der einzige Weg, wie die beiden Funktionen gleichzeitig störungs- und redundanzfrei verwendet werden können. So kann der Benutzer nach Bedarf problemlos entweder über USB oder über Bluetooth mit der Gegenstelle kommunizieren.

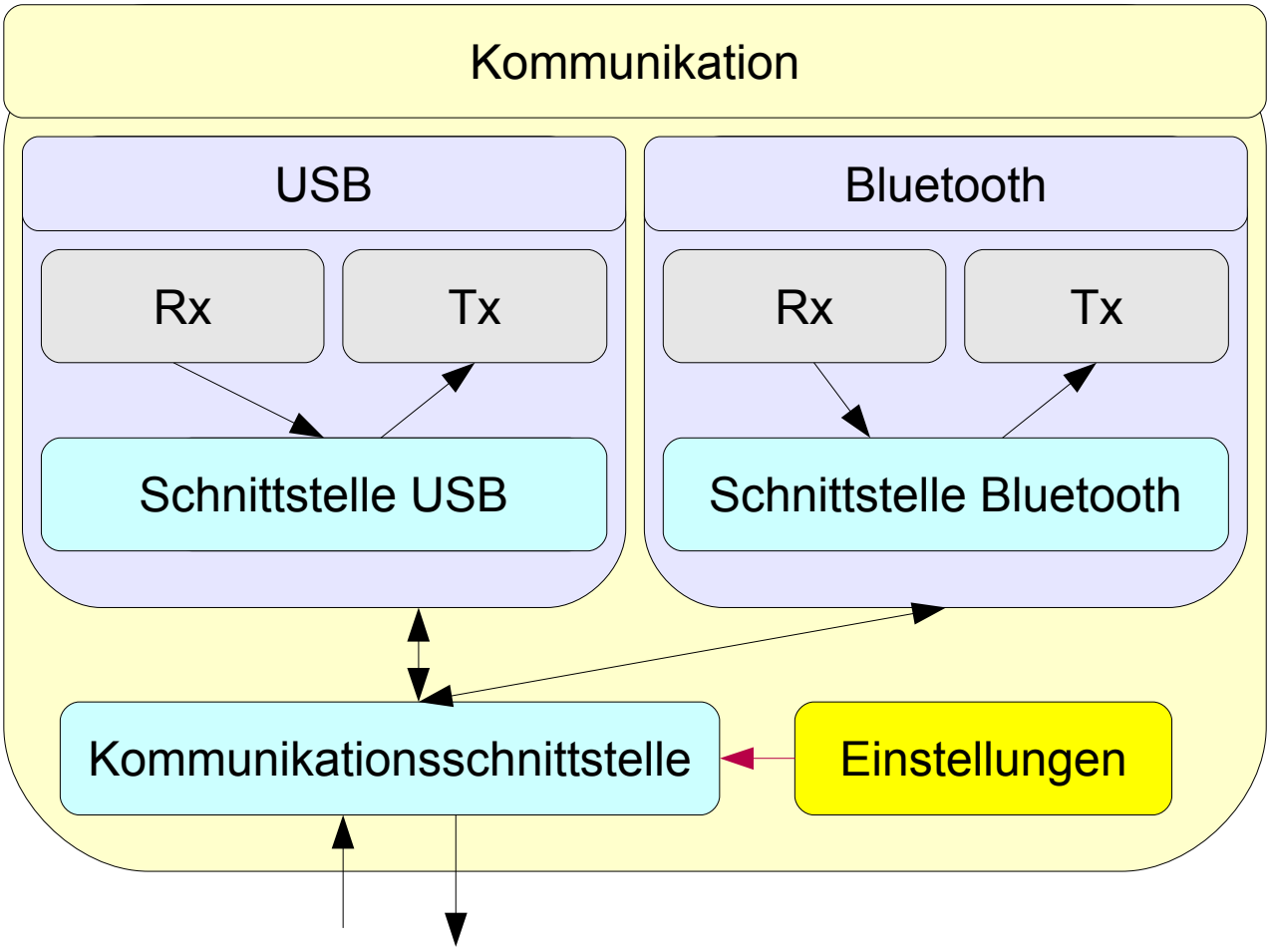

Abbildung 29: Kombinierte Spezialisierung der Kommunikationsschnittstelle

3.4 Struktur einer Anwendung

Der vollständige Entwurf der Anwendung kann dann, auf entsprechenden Abstraktionsstufen, wie in der Abbildung 30 aussehen. Es ist zu beachten, dass es gewollt war, einige Details, die im Entwurf vorhanden waren in dieser Sicht auszublenden. Durch einen stufenweisen Einstieg in den Entwurf wird es für den Betrachter deutlich einfacher die Einzelheiten des Entwurfs zu verstehen. Falls er bestimmte Aspekte des Entwurfs genauer kennenlernen möchte, sollte dies über weiterführende, spezialisierte Ansichten erfolgen. Deswegen ist immer darauf zu achten, welche Informationen dem Betrachter zu einem gewissen Zeitpunkt präsentiert werden sollten - es sollte genug sein um die Abstraktionsstufen zu verstehen, aber nicht zu viel um ihm mit unnötigen Details zu verwirren oder zu überfordern.

Letztendlich hat die Anwendung, die nach diesem strukturierten Entwurf erstellt wurde unter anderem folgende Vorteile:

- Die Steuerung der Anwendung kann ohne Änderungen entweder vom Benutzer, oder von außerhalb über die Kommunikationsschnittstelle erfolgen.
- Neue Kommunikationswege können in das System relativ leicht und zeitsparend eingepflegt werden.
- Durch die Hardwareabstraktionsschicht sowie die Datenabstraktion ist die Austauschbarkeit der Sensoren sowie Aktoren über eine Anpassung oder einen Austausch dieser Schichten möglich.
- Durch die Datenabstraktion ist es möglich gleichzeitig unterschiedliche Sensoren für gleiche Aufgaben zu verwenden (z.b. 2 Typ verschiedene Temperatursensoren) es muss dafür nur ein entsprechender Sensortreiber in dieser Schicht implementiert werden.
- Ganze Teile der Anwendung könnten bei Bedarf in separate Anwendungen ausgelagert werden z.B. die Steuerung sowie Datenabstraktion und Hardwareabstraktion auf einen Prozessrechner, die Datenspeicherung gleichzeitig auf einen dedizierten Datenbankrechner.

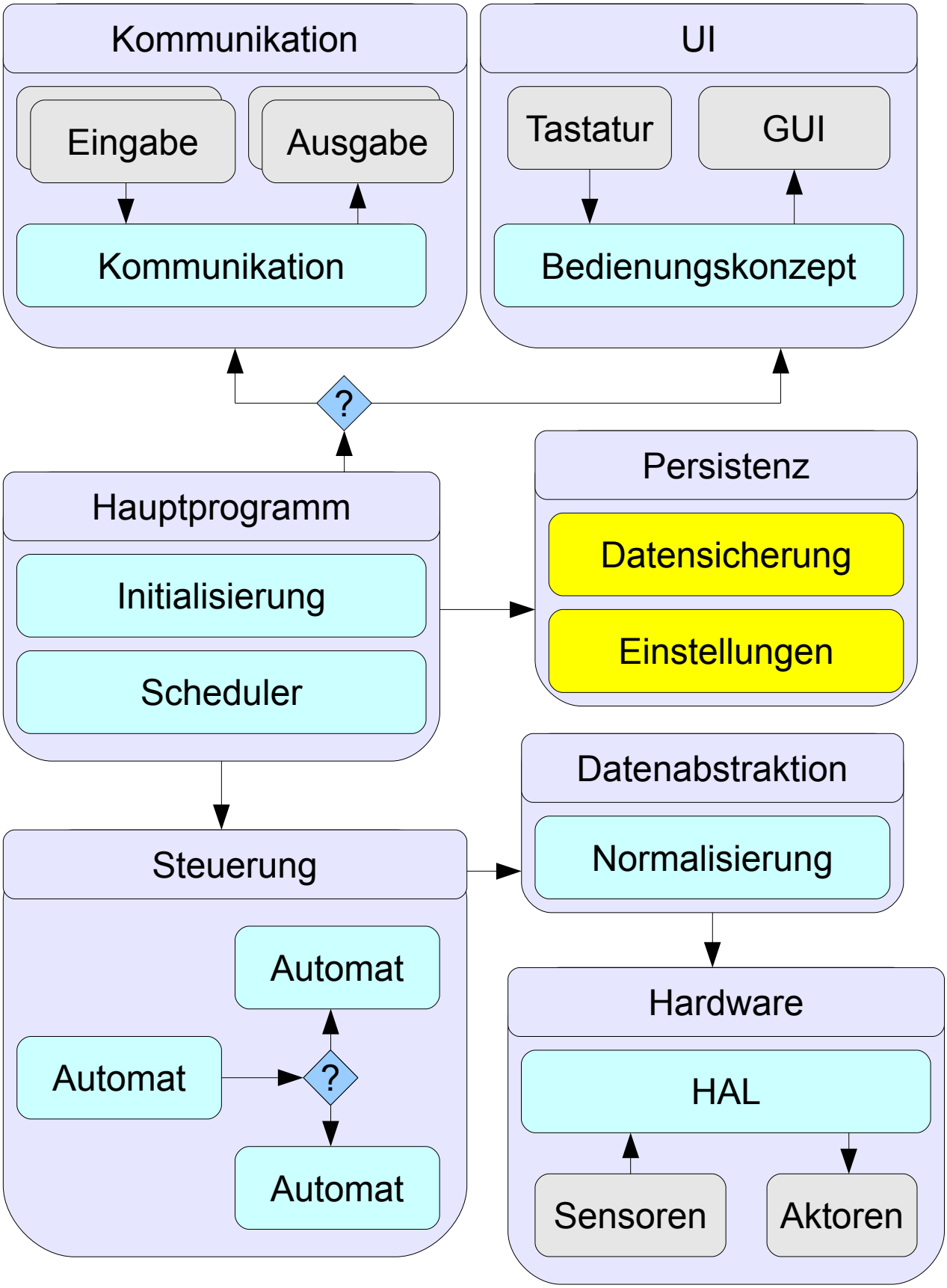

Abbildung 30: Strukturierte Anwendung

3.5 Abgeleitete Anwendungen

Nach einer gewissen Zeit, möchte der Kunde eventuell seine Anwendung von einem Fernarbeitsplatz verwenden. Wurden die Hausaufgaben beim Entwurf, sowie bei der folgenden Implementierung entsprechend gut gemacht, lässt sich aus den vorhandenen Komponenten relativ „schnell“ eine zweite Anwendung ableiten (Abbildung 31). Die UI kann unter Umständen vollständig übernomen werden. Die Kommunikation ist mit hoher Wahrscheinlichkeit anzupassen, da diese jetzt in die andere Richtung abläuft. Das Hauptprogramm muss mit Sicherheit neu geschrieben werden, aber auch hier können einige Teile (Submodule) vermutlich wiederverwendet werden. Somit sinkt die dafür notwendige Entwicklungszeit enorm. Ein zusätzlicher, vermutlich noch wichtiger, positiver Effekt ist, dass Erweiterungen der UI (sowie anderen wiederverwendeten Funktionalitäten) sofort in beiden Anwendungen vorhanden sind. Auch eventuell vorhandene Fehler sind nur einmal zu beseitigen.

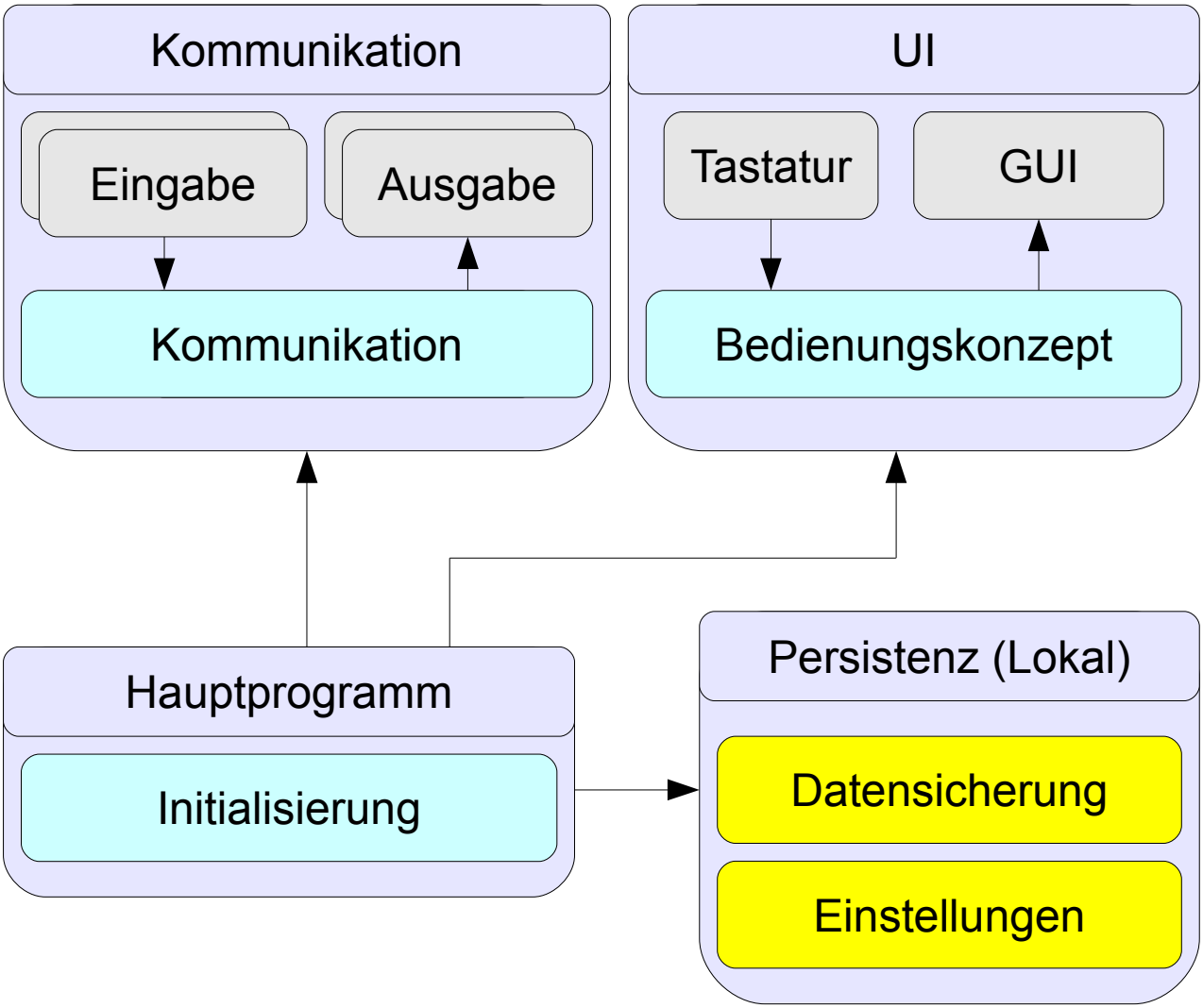

Abbildung 31: Struktur einer Abgeleiteten Anwendung

3.6 Weitere Beispiele aus der Robotik

3.6.1 Modellierung eines Roboterverhaltens

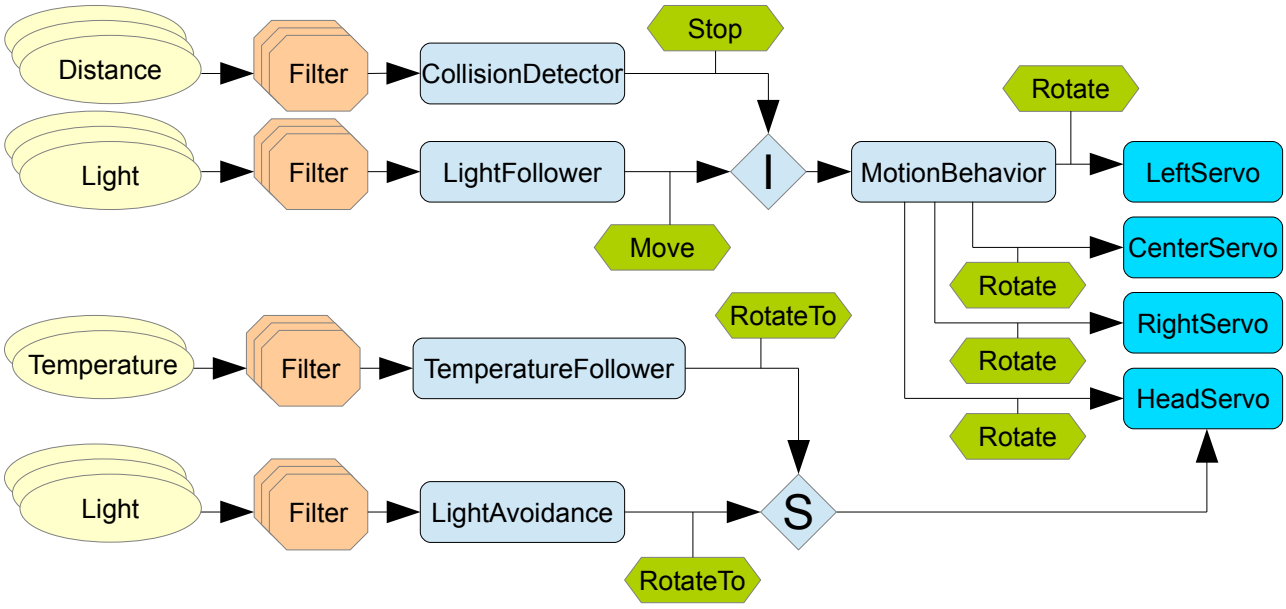

Abbildung 32: Verhaltensstruktur eines Roboters

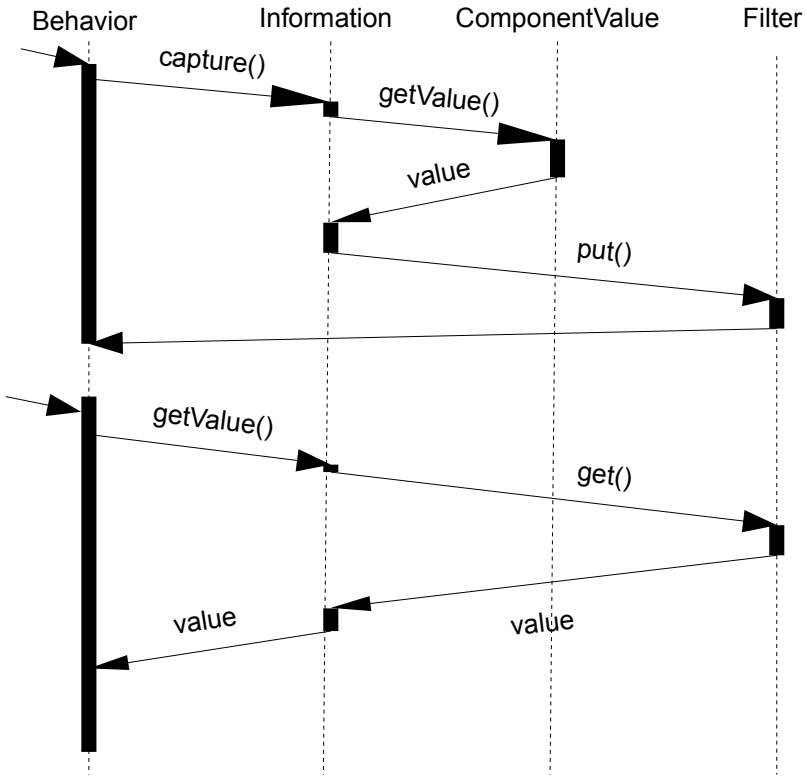

Abbildung 33: Informationsgewinnung eines Verhaltens

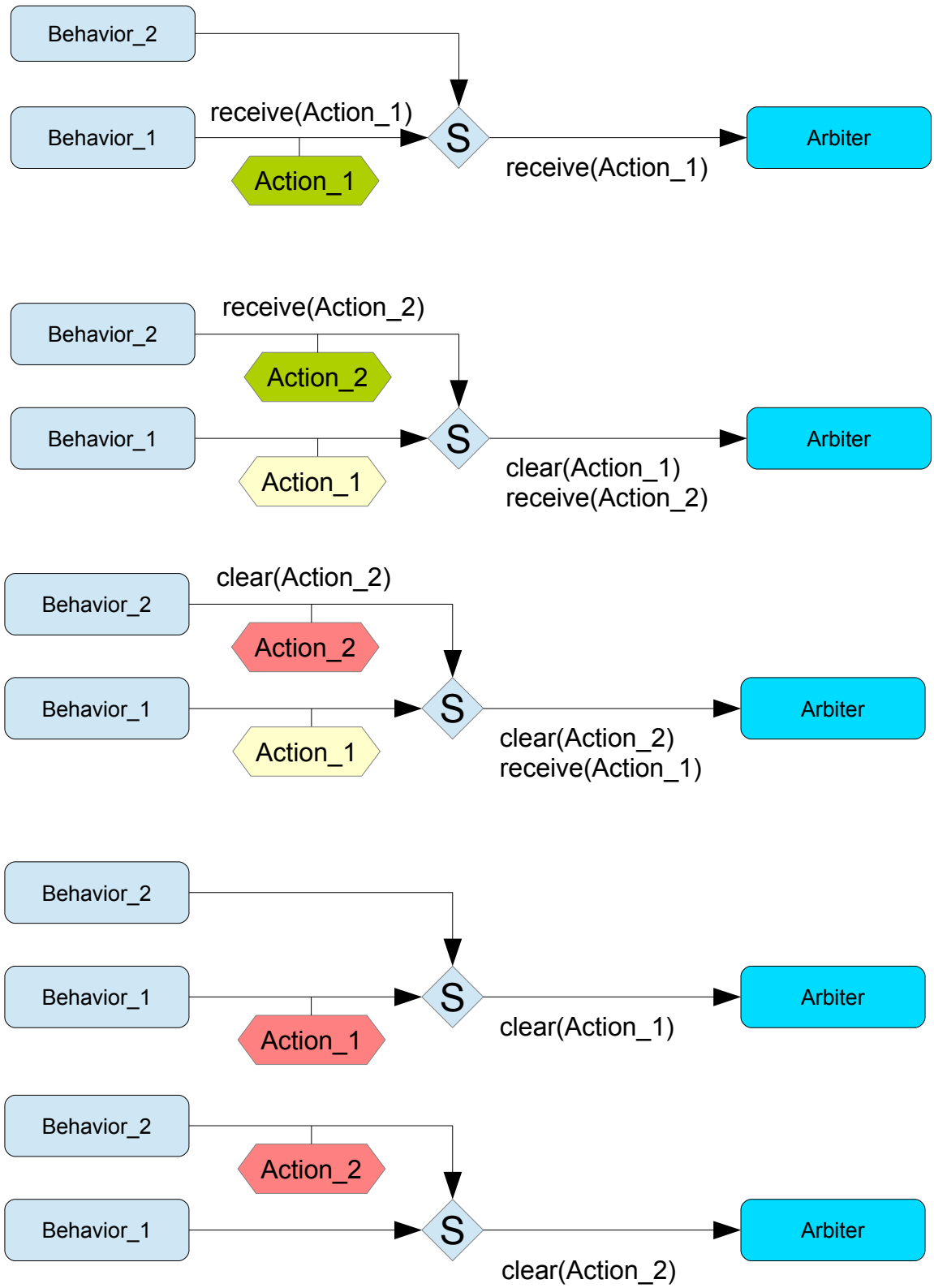

Abbildung 34: Verhalten einer Supressornode

3.6.1 Robotiksystem

Als weiteres Beispiel schauen wir uns mal ein Robotiksystem an , das zuerst aus der Logik eines Roboters besteht, der in Java implementiert wurde. Dieser logische Teil kommuniziert mit dem echten Roboter , oder dessen Simulation über eine eigene Kommunikationsschnittelle (USB, Serial, Bluetooth, Ethernet, interner Datenaustausch) entweder mit einem echtem Roboter (Abbildung 36) oder einer Simulation (Abbildung 35). Im späteren Verlauf des Projekts wurden über die Agenten auch Fremdroboter angeschlossen, da es nicht möglich war deren Firmware entsprechend anzupassen. Deswegen wurde die Fremdhardware über modifizierte Agenten angesteuert.

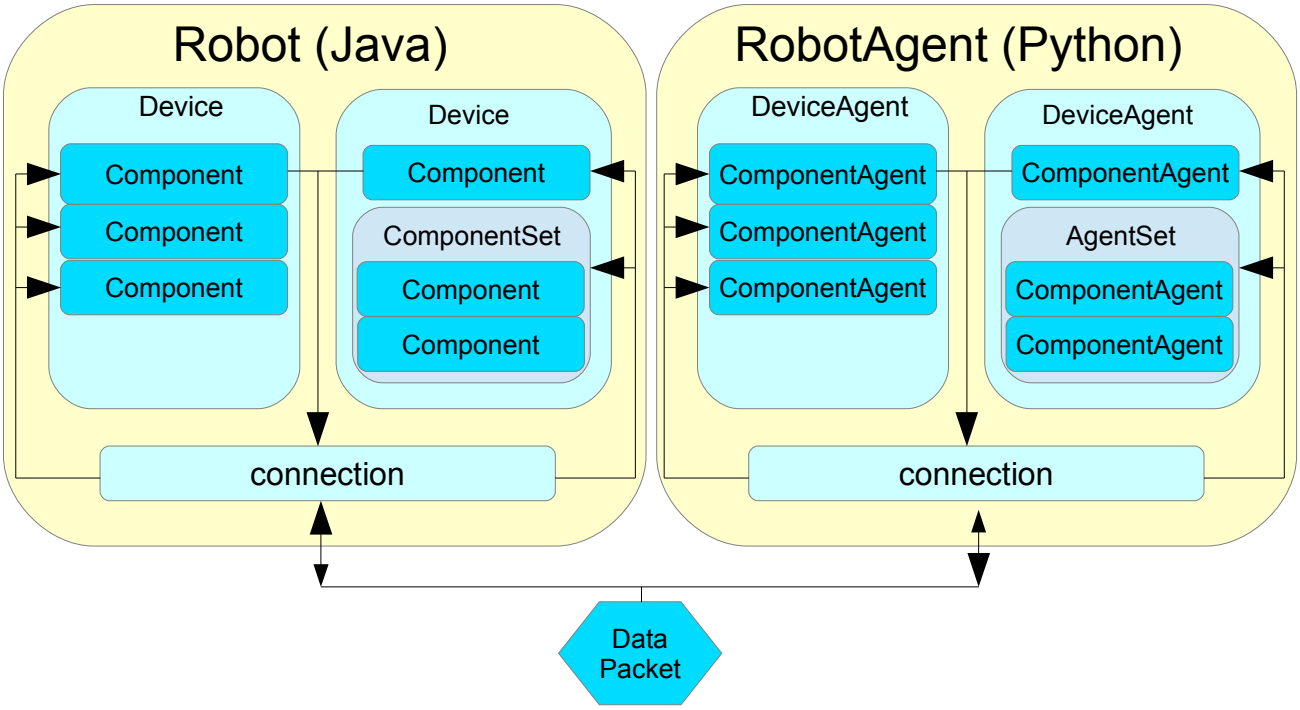

Abbildung 35: Robotiksystem Roboter (JAVA) mit Simulator(Python)

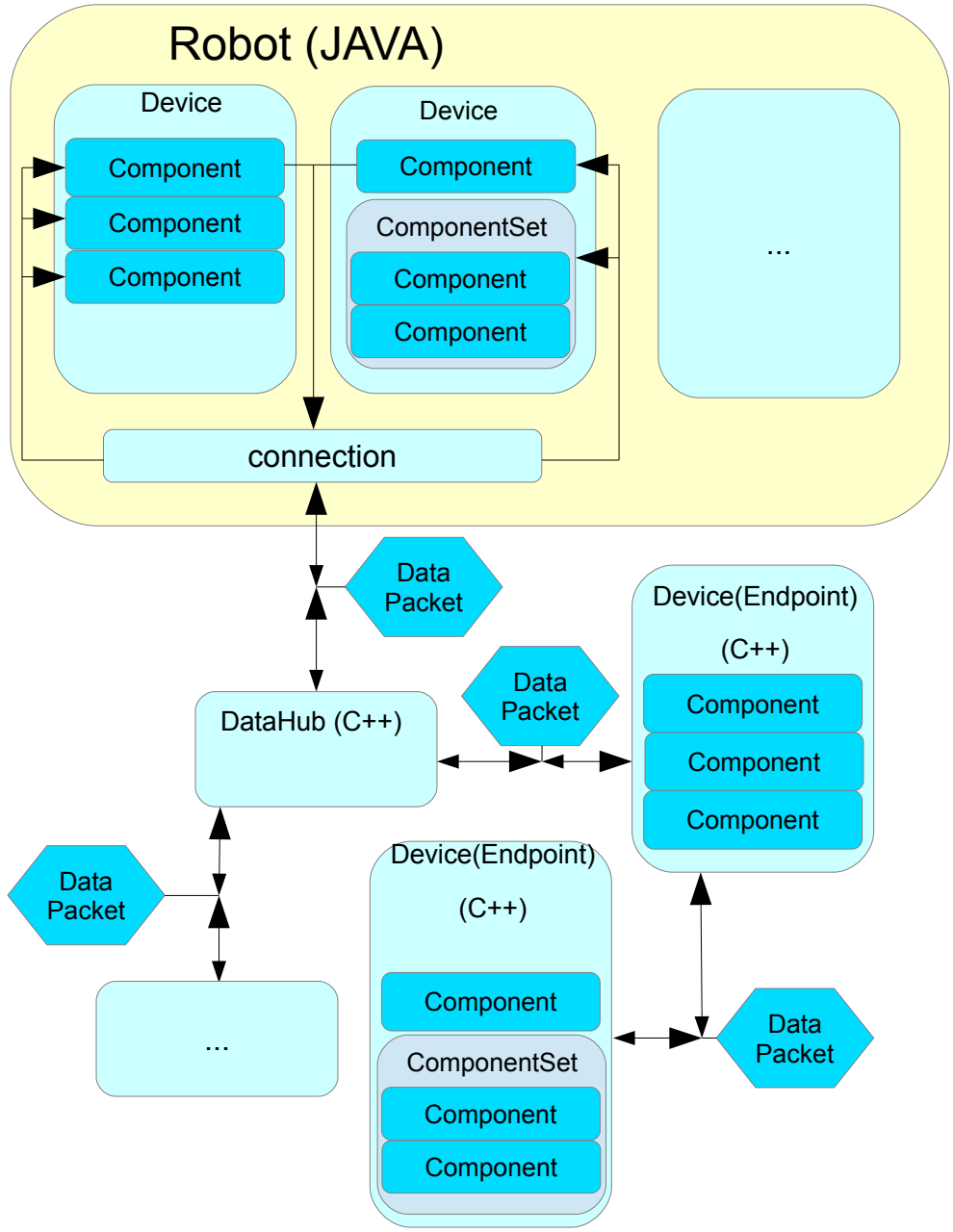

Abbildung 36: Robotiksystem mit einem externem Hardwareroboter

3.6.1 Starten eines Spieles (Aufgabe SoSe2017)

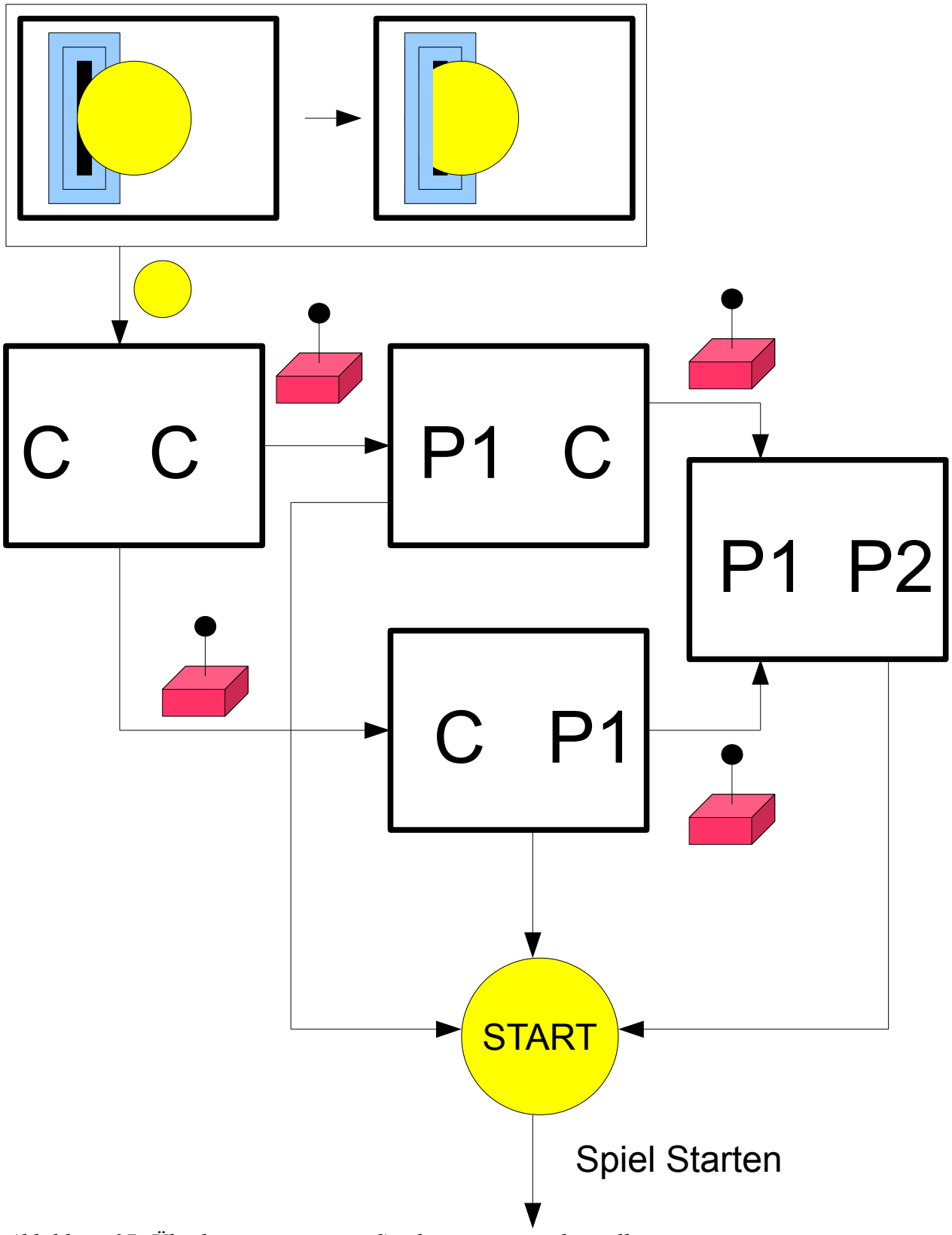

Abbildung 37: Überlegung zu wie ein Spiel gestartet werden soll.

3.6.2 Entwurf bei Spielgestaltung

Bei der Konzeption und Entwicklung eines MMO Spiels ist einer der wichtigsten Faktoren für die spätere Beliebtheit (lese € & $) des Spiels die Gegebenheit, dass die Fähigkeiten der Spieler nicht gleichmäßig verteilt sind (Abbildung 38). Eine kleine Gruppe der Spieler, die sogenannten . Unicums bzw. Super unicums verderben den restlichen Spielern das Spielerlebnis, so dass diese das Spiel letztendlich als „unfair“ wahrnehmen. Deswegen sollte bereits in einem guten Spielentwurf beachtet werden, wie die Auswirkung der (Un-) Fähigkeiten der einzelnen Spieler durch Spieldesign ausgeglichen werden kann.

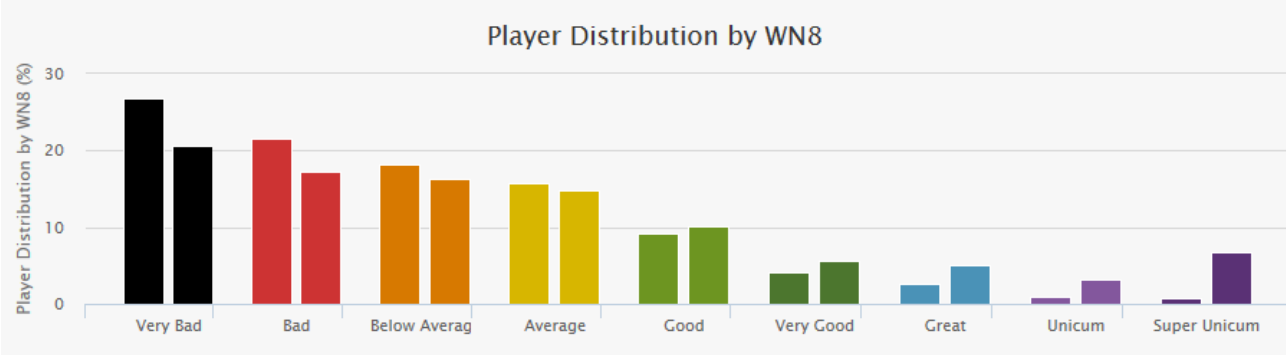

Abbildung 38: Verteilung der Spieler nach Spielleistung bei „Word of Tanks“ Quelle Wot-life.com 9.5.2017

4 Beispielaufgabe

4.1 Aufgabenstellung

Ein Physiker stellt folgende Aufgabe:
Entwickeln Sie eine autonome Temperaturregelung. Die zu haltende Temperatur soll über ein Display eingestellt werden können. Zusätzlich soll in der Regelung ein Modus vorgesehen werden, bei dem die Wassertemperatur so nahe wie möglich am Siede- oder Schmelzpunkt gehalten wird.

4.2 Überlegungen

Bei der Analyse der Aufgabenstellung wird zuerst festgestellt, dass die Aufgabestellung aus der Sicht des Entwicklers stellenweise unvollständig ist. Nach einer grundsätzlichen Analyse stellt der Entwickler folgendes fest:

1) Es fehlen vollständig die Angaben auf welcher Hardware die Software ausgeführt werden soll, da es autonom ist, könnte angenommen werden, dass es sich um einen Mikrocontroller handelt. Da jegliche Annahmen schlecht sind, wird an der Stelle nachgefragt.
2) Die Eingabe der Temperatur über das Touch Display wäre zwar technisch realisierbar, aber ist es wirklich das, was sich der Auftraggeber wünscht?
3) Wie genau ist „so nahe wie möglich“, bzw. wie genau soll die sonstige Regelung sein?
4) Was soll auf dem Display, außer der eingestellten Temperatur, sonst noch dargestellt werden?
5) Was darf die Hardware den insgesamt kosten?
6) Temperaturangaben in Kelvin oder Celsius ?

Mit den gesammelten Fragen wird jetzt der Auftraggeber konfrontiert,dabei wird folgendes festgestellt:

1) Es stellt sich heraus, dass der Aufgabensteller tatsächlich daran dachte die Software auf einem Mikrocontroller, dem Raspberry Pi, ablaufen zu lassen. Dort kann er auch noch gleich seinen Web/Medienserver laufen lassen.
2) Nein er will keinen Touch Display haben, er hätte schon gerne ein paar Tasten, sowie einen Drehregler (lese Potentiometer).
3) So gut wie man es messen kann (tolle Antwort).
4) Aktuelle Temperatur, eingestellte Temperatur, grafischen Temperaturverlauf für die letzten 10 Tage mit farbiger Markierung der Temperaturausreißer.
5) Am besten nichts.
6) Kelvin.

Nach der Rücksprache mit dem Auftraggeber stellt der Entwickler weitere Nachforschungen an, dabei stellt er fest:

Von der Aufgabe zur Anwendung

1. Mit dem Raspberry Pi sind eigentlich keine zeitlich exakten Messungen, geschweige denn zeitkritische Regelungen möglich. Aus diesem Grund wird dem Auftraggeber ein 8Bit µC von Atmel (ATMEGA32U4) vorgeschlagen. Des Weiteren ist die Erfassung der Signale des Drehreglers am Raspberry Pi problematisch. (Vorschlag wird widerwillig angenommen).
2. Als Eingabegeräte werden vorgeschlagen: eine „Siedepunkt-„ , eine „Schmelzpunkt-“, sowie eine „Ok- Taste“, und „Abbruch- Taste“, eine numerische Tastatur zu direkten Temperatureingabe, sowie zwei Potentiometer (Grob- und Feineinstellung). (Vorschlag wird angenommen).
3. Siehe 5.
4. Da gibt es bereits kleine TFT Displays, teilweise sogar mit entsprechenden Bibliotheken zum Zeichnen der benötigten Bildschirminhalte. Verwendung mit einem kleinen Mikrocontroller ist schwierig, da er wenig Speicher hat. Vorschlag – Die UI auf dem Pi auszulagern (es wird zu Teuer kein Pi).
   (a) Wie sind die Temperaturausreißer definiert? (Definition des Kunden +- 50%)
5. Es wird eine Übersicht mit Sensoren, sowie deren Kenndaten und Preise verschiedener Hersteller erstellt. Diese wird dem Kunden vorgelegt. (Kunde sucht sich an Hand der Kenndaten einen Sensor aus).

Analyse fertig, oder?

STOP

An dieser Stelle hat der Auftragnehmer einen entscheidenden Fehler gemacht. So kann die Regelung mit großer Sicherheit nicht den Anforderungen des Physikers entsprechen, warum?

4.3 Kontext des Auftraggebers

Wie wir vorher festgestellt haben, ist der Auftraggeber ein Physiker. Es ist eher unwahrscheinlich, dass dieser mit Ihrer Regelung seinen allmorgendlichen Kaffee kochen bzw. warmhalten will. Die Vermutung liegt sehr nahe, dass er damit wohl irgendwelche Experimente durchführen will. Vor allem die Einstellungsmöglichkeit „Siedepunkt“ sollten die Alarmglocken läuten lassen. Der Siedepunkt ist bei einem Informatiker als eine Konstante WASSER_SIEDEPUNKT = 100° definiert. Das Problem liegt hier nicht daran, dass der Siedepunkt eigentlich bei 99,97°C liegt, sondern, dass es sich hier um einen ganz besonderen Betrachtungsfall handelt, nämlich bei einem Druck von exakt 1atm. Allerdings weiß bereits ein Bergsteiger, dass schon auf einem 3000m hohen Gipfel der Siedepunkt bei ca. 90°C liegt. Hätte der Informatiker sich den Versuchsaufbau des Physikers vorzeigen lassen, wäre ihm vermutlich der riesige Druckkessel aufgefallen, in dem die Temperaturregelung stattfinden sollte. Die Frage was der Physiker sonst noch ins Wasser kippen könnte, um den Schmelzpunkt zu beeinflussen lassen wir mal außen vor....

4.4 Aufgabenstellung

Es ist ein Mikrocontroller basiertes System zur Temperatursteuerung zu entwickeln. Dabei sollte das System zwei verschiedene Arbeitsmodi beherrschen. Einerseits sollte eine Regelung auf eine manuell voreingestellte Temperatur realisiert werden, anderseits soll es möglich sein die Regelung auf Schmelz- oder Siedepunkt des Wassers einzustellen. Die für die Regelung des Siedepunktes notwendige Luftdruckinformation soll über einen Luftdrucksensor ermittelt werden. Die Eingabe der Steuerwerte erfolgt entweder über eine numerische Tastatur, oder über zwei Potentiometer. Die Umschaltung der Arbeitsmodi sollte über Tasten realisiert werden. Die Ausgabe der eingestellten, sowie der aktuellen Temperatur soll über ein Grafisches Display erfolgen. Neben den diskreten Temperaturangaben soll noch zusätzlich ein Temperaturverlauf der letzten 10 Tage als Diagramm dargestellt werden. Dabei sind die vorgekommenen Temperaturausreißer rot zu markieren. Als Temperaturausreißer sind Werte mit +-50% Unterschied zum Vorgängerwert definiert.

Nochmal ein Hinweis – da das System die diese Aufgabenstellung beschreibt noch nicht existiert, wird es in „soll“ Form und nicht in „ist“ (bereits vorhanden) oder „wird“ (bereits in Entwicklung oder übliche Vorgehensweise) beschrieben.

4.5 Analyse

Im Rahmen des Projektes „Temperatur Versuchsstrecke“ ist für den Kunden eine autonome Regelung der Temperatur zu realisieren. Diese sollte außer der Regelung auf die manuell eingestellte Temperatur auch die Regelung auf den Siede- sowie den Schmelzpunkt des Wassers beherrschen. Da die korrekte Regelung des Siede- sowie Schmelzpunktes nur mit einer Kombination aus Temperatur- sowie Drucksensor zu bewerkstelligen ist, wird aus diesem Grund

zusätzlich zum Temperatursensor noch ein Drucksensor benötigt.

Für die Temperaturmessung ist ein Thermoelement als Sensor notwendig. Dieses gibt es als J (0°C...750°C), K (-200°C...1250°C), E (-200°C...900°C) sowie T Typen (-250°C...350°C). Der J Typ scheidet von vornherein als ungeeignet aus, da dessen Temperaturbereich nicht unterhalb 0°C definiert ist. Für die restlichen drei Sensortypen wurde dessen Genauigkeit als zweites Kriterium hinzugenommen. Auf den ersten Blick schien der Typ E wegen seiner kleineren Toleranz am besten geeignet zu sein. Da dieser aber eine höhere Messbandbreite (1100°C gegenüber 600°C des T Typs) aufweist, ist seine absolute Genauigkeit kleiner als die des T Typs. Somit ist ein Thermoelement vom T Typ zu bevorzugen.

Typ	Temperaturbereich	Fehltoleranzen
J	0°C...750°C	0,75% mindestens 2,2°C
K	-200°C...1250°C	0,75% mindestens 2,2°C
E	-200°C...900°C	0,5% mindestens 2,2°C
T	-250°C...350°C	0,75% mindestens 1,0°C

Tabelle 3 Kenndaten der Thermoelemente

Der Druckmessumformer WIKA Typ S-20 erfüllt die von den Kunden vorgegebene Messgenauigkeit und ist durch die IP67 Schutzklasse für entsprechende Anwendungen zugelassen.

Da eine eigene Implementierung des PID Reglers zur Temperaturregelung zu aufwendig ist, soll statt einer Softwarelösung der Wachendorff UR3274U6 PID Regler verwendet werden.

Für die Ausgabe sowie Menüführung ist ein Waveshare 320x240 Pixel 2.8 Zoll LCD Display ausreichend. Die Ansteuerung des Displays kann über Softwarebibliothek des Herstellers erfolgen.

Um die Pins des µControllers zu sparen sollten alle Tasten über Schieberegister (47HC165) angeschlossen werden.

Des weiteren ist ein Bedienungskonzept für das Messsystem zu entwerfen und der Entwurf entsprechend zu implementieren. Dabei sind zwei Potentiometer für die Eingabe der Solltemperatur in 0.1°K Schritten vorgesehen. Der erste Potentiometer stellt den groben Temperaturbereich ein, die Feinjustierung der Temperatur erfolgt über den zweiten Potentiometer. Der zweite Weg zur Temperatureingabe erfolgt über eine numerische Tastatur. Des Weiteren sind noch zwei Tasten für die automatische Einstellung des Siede- sowie des Schmelzpunkts als feste Regelpunkte, eine „OK“ Taste (zur Festlegung der aktuellen Regeltemperatur) sowie eine „Abbruch“ Taste (Rücksetzten der Anzeige auf aktuellen Regelpunkt) zu verwenden. Der aktuelle Arbeitsmodus soll über einen Text

auf dem Display gekennzeichnet werden. Neben der eingestellten Temperatur ist auch die aktuelle Temperatur über das Display anzuzeigen. Alle Temperaturangaben sind in Kelvin vorzunehmen.