
Projekt Softwaretechnik Sommersemester 2016

im Studiengang Softwaretechnik u. Medieninformatik
der Fakultät Informationstechnik



Flow Design

Thema: Erstellen eines Modeling Tool Prototyps für Flow Design

Team:	Daniel Glinka	<daglit01@hs-esslingen.de>	749512
	Daniel Kratzel	<dakrit03@hs-esslingen.de>	749318
	Johannes Schlier	<joscit13@hs-esslingen.de>	749815
	Johannes Steinhülb	<jostit01@hs-esslingen.de>	749818

Betreuer: Prof. Dr.-Ing. Reinhard Schmidt

Firma: IT-Designers GmbH

*„Abhängigkeiten sind eine der größten Geißeln
der Softwareentwicklung.“*

Ralf Westphal

Inhaltsverzeichnis

1 Einleitung	1
1.1 Flow Design	1
1.1.1 Prinzipien	1
1.1.2 Diagramme	3
2 Spezifikation	6
2.1 Einleitung	6
2.1.1 Zweck der Spezifikation	6
2.1.2 Leserkreis der Spezifikation	6
2.1.3 Einsatzbereich und Ziele von Flow Design	7
2.1.4 Verwendete Abkürzungen und Schreibweisen	7
2.2 Allgemeine Beschreibung	7
2.2.1 Einbettung	7
2.2.2 Überblick über die Funktionen	7
2.2.3 Mengengerüst	8
2.2.4 Profil der Flow Design Nutzer	8
2.2.5 Entwicklungswerkzeuge	8
2.3 Einzelanforderung	9
2.3.1 Nicht-Funktionale Anforderungen	9
2.3.2 Funktionale Anforderungen	10
2.4 Anwendungsfälle	11
2.4.1 Anwendungsfalldiagramm	11
2.4.2 Usability Patterns	11
2.4.3 Kontextdiagramm	12
2.5 Anhang	13
2.5.1 Begriffslexikon	13
3 Projektplan	14
3.1 Einleitung	14
3.1.1 Zweck des Projektplans	14
3.1.2 Projektüberblick	14
3.1.3 Entwicklungsphilosophie	14
3.1.4 Vertragliche Grundlagen	14
3.2 Risiken	15
3.2.1 Risiken, ihre Bewertung und Gegenmaßnahmen	15
3.3 Entwicklungsplan	16
3.3.1 Zeitplan und Meilensteine	16
3.4 Entwicklungsprozess	16
3.4.1 Dokumentation	16
3.4.2 Qualitätssicherung	16
3.5 Projektorganisation	17

4 Benutzeroberfläche	18
4.1 Einleitung	18
4.1.1 Erste Ideen	18
4.1.2 Konkrete Entwürfe	20
4.1.3 Grafischer Prototyp	25
4.1.4 Finale Oberfläche	30
4.1.5 Farbschema	33
5 Umsetzung	35
5.1 Allgemein	35
5.2 Projektplan	35
5.3 Problemanalyse	35
5.4 Spezifikation	37
5.5 Systementwurf	37
5.5.1 Model-Layer	38
5.5.2 Data Access-Layer	40
5.5.3 Service-Layer	41
5.5.4 UI-Layer	42
5.6 Implementierung	44
5.6.1 Datenspeicherung	45
5.7 Test	46
5.8 Versionsverwaltung	46
5.9 Zeitmanagement	47
6 Produkt	48
6.1 User-Guide	48
6.1.1 Einleitung	48
6.1.2 Erstellen eines neuen Projekts	48
6.1.3 Speichern eines Projektes	51
6.1.4 Öffnen eines vorhandenen Projekts	51
6.1.5 Erstellen eines Diagramms	54
6.1.6 Diagramm bearbeiten	57
6.1.7 Diagramm löschen	60
6.2 Code Beispiel	60
6.2.1 Schritt 1: Model erstellen	60
6.2.2 Schritt 2: Anpassung des Component Service	62
6.2.3 Schritt 3: Erstellung des ViewModels	64
6.2.4 Schritt 4: Festlung der Darstellung	67
6.2.5 Ergebnis	68
6.3 Statistiken	68
7 Schlusswort	74
Literatur	75

Abbildungsverzeichnis

1	Beispiel Mutual Oblivion	2
2	Integration Operation Segregation	2
3	Beispiel System-Umwelt-Diagramm	3
4	Beispiel Maskenprototyp	4
5	Beispiel Flow Diagramm	5
6	Anwendungsfalldiagramm	11
7	Kontextdiagramm	12
8	System-Umwelt Diagramm Ressourcen	18
9	System-Umwelt Diagramm	19
10	Maskenprototyp	19
11	Flow Design	20
12	Startseite	21
13	Projekt erstellen (Option 1)	21
14	Projekt erstellen (Option 2)	22
15	Sytem Umwelt	22
16	Maskenprototyp wählen	23
17	Maskenprototyp	23
18	Flow Diagramm	24
19	Flow Diagramm Detailansicht	24
20	Prototyp - Startseite	25
21	Prototyp - Projekt erstellen	26
22	Prototyp - System Umwelt Diagramm	27
23	Prototyp - Maskenprototyp erstellen	28
24	Prototyp - Maskenprototyp	28
25	Prototyp - Flow Diagramm	29
26	Prototyp - Flow Diagramm - nächste Ebene	30
27	Startseite	31
28	System Umwelt Diagramm	31
29	Maskenprototyp	32
30	Flow Diagramm	32
31	Oberflächenausschnitt in hellem Farbschema	33
32	Oberflächenausschnitt in dunklem Farbschema	33
33	Farbpalette des hellen Farbschemas	34
34	Farbpalette des dunklen Farbschemas	34
35	Problemanalyse Flow Diagramm	36
36	Problemanalyse beim Kunden am Flipchart	36
37	Layered Architecture	38
38	Hierarchie der Diagramme	39
39	Komponenten des System-Umwelt-Diagramms	39
40	Aufbau des Data Access-Layers	40
41	Teilklassendiagramm des Service Layers	41

42	Fenster mit Startseite	42
43	Startseite mit Infoseite	43
44	Startseite mit Projekterstellung	43
45	Kanban Board Trello	44
46	Kanban Board Detail	45
47	Schritt 1: „Neues Projekt“ auswählen.	48
48	Schritt 2: Dem Projekt ein Namen geben.	49
49	Schritt 3: Über „durchsuchen“ wird ein Ordner zum Speichern ausgewählt und mit „OK“ bestätigt.	49
50	Schritt 4: Über „erstellen“ wird das Projekt angelegt.	50
51	Schritt 5: In der Statusleiste wird nach erfolgreichem Erstellen des Projekts dessen Name und Speicherort angegeben.	50
52	Schritt 1: Um die Änderungen an dem Aktuellen Projekt manuell zu speichern, klickt man in der Menüleiste auf das „Speichern“ Symbol.	51
53	Schritt 1: Aus der Startseite heraus können über mehrere Schaltflächen Projekte geöffnet werden. Klickt man auf eines der Projekte in der Liste der zuletzt verwendeten Projekte, wird es direkt geöffnet. Über die „Projekt öffnen“ Schaltflächen Links und im Menü am oberen Rand, muss ein Projekt von der Festplatte ausgewählt werden.	52
54	Schritt 5: Aus der Diagrammansicht kann die „Projekt öffnen“ Schaltfläche verwendet werden.	53
55	Schritt 3: Um ein Projekt von der Festplatte zu laden, muss man in dessen Speicherpfad gehen, die „flow“ Datei auswählen und öffnen.	54
56	Schritt 1: Über einen der drei Diagramm Buttons das Dropdown-Menü öffnen.	55
57	Schritt 2: Einen Namen eingeben und auf „Erstellen“ klicken.	55
58	Schritt 3: Das Diagramm wurde erstellt und kann nun ausgewählt werden.	56
59	Schritt 4: Die Ansicht zum Bearbeiten des Diagramms.	56
60	Schritt 1: Eine Komponente von links per Drag and Drop in das Diagramm Feld ziehen.	57
61	Schritt 2: Es können nun weitere Komponenten hinzugefügt werden (z.B. eine Ressource).	58
62	Schritt 3: Klickt man auf einen Konnektor, kann durch gedrückt halten der linken Maustaste und Ziehen auf einen weiteren Konnektor eine Abhängigkeit auf ein Anderes System hinzugefügt werden. Das System von dem man die Abhängigkeit aus zieht, ist von dem Ziel abhängig.	58
63	Schritt 4: Auf gleiche Weise können auch Aktoren hinzugefügt werden. Diese sind im Normalfall vom System abhängig, also werden die Verbinden vom Aktor aus gezogen.	59
64	Schritt 5: Um eine Abhängigkeit zu Löschen, klickt man mit der Rechten Maustaste auf sie.	59
65	Schritt 1: Um ein Diagramm innerhalb eines Projekts zu löschen, klickt man in dem Entsprechendem Diagramm Auswahlmenü auf das „X“ rechts neben dem zu löschenen Diagramm.	60
66	Fertige Komponente	68

67	NDepend Metriken Allgemein	69
68	NDepend Metriken Abhangigkeiten	70
69	NDepend Metriken Abstractness vs. Instability	71

1 Einleitung

Saubere Softwareentwicklung braucht eine Methode. Prinzipien wie Single Responsibility und Separation of Concerns alleine reichen nicht aus, um sauberen Code mit möglichst wenigen Abhängigkeiten zu schreiben. Ralf Westphal, Mitgründer der Initiative „Clean Code Developer“, beschäftigt sich seit einigen Jahren mit der Entwicklung einer solchen Methode, welche er „Flow Design“ nennt und in seinem Buch „The Architect’s Napkin“[2] beschreibt.

Mit dem Entwurfskonzept „Flow Design“ hat er einen leichtgewichtigen und sehr praktikablen Weg gefunden, den jeder Entwickler jederzeit ohne großen Aufwand nutzen kann, um Software zu entwerfen. Flow Design hilft bei der Vermeidung von Abhängigkeiten durch eine Entkopplung der Module im System. Erreicht wird dies durch die Festlegung von Schnittstellen lediglich in Form von Datenflüsse, sowie durch die Definition und strukturierter Verdrahtung eigenständiger Funktionseinheiten.

Ziel dieses Projektes ist der Prototyp eines Modeling Tools für das Erstellen von Diagrammen, welche auf der Vorgehensweise „Flow Design“ von Ralf Westphal basieren.

1.1 Flow Design

1.1.1 Prinzipien

Flow Design basiert auf dem „Single Responsibility Principle“ nach Robert C. Martin. Jede Funktionseinheit (Methode, Klasse, etc.) soll jeweils nur einer einzigen Aufgabe nachgehen. Ralf Westphal erweitert diese Entwurfsrichtlinie mit der Einführung des „Principle of Mutual Oblivion“ und des „Integration Operation Segregation Principle“.

- Principle of Mutual Oblivion

Das „Principle of Mutual Oblivion“ besagt, dass jede Funktionseinheit nichts von seiner Umwelt wissen soll. Beim Entwurf einer Funktion oder Methode geht man gerne von verschiedenen Umweltzuständen oder ähnlichem aus, welche das Verhalten der zu entwerfenden Funktion beeinflussen.

Genau davon sollte man sich jedoch entfernen. Funktionen sollten so entworfen werden, dass sie, wenn sie die richtigen Parameter bekommen, das tun, was erwartet ist. Sie sollten also funktionieren, egal in welchem Programm oder in welchem Umfeld sie eingesetzt werden.

Sie sind damit unwissend, was alle anderen Funktionen angeht. Umgekehrt sollten auch alle anderen Funktionen nichts über die Funktion wissen, um solche

Annahmen zu eliminieren („Mutual Oblivion“).

Dadurch wird die Wiederverwendbarkeit von diesen Funktionen oder Modulen verbessert. Auch die Wartung wird leichter. Da die Funktion nicht von äußeren Werten oder Ähnlichem abhängig ist, muss man wirklich nur die Funktion an sich anschauen und ändern.

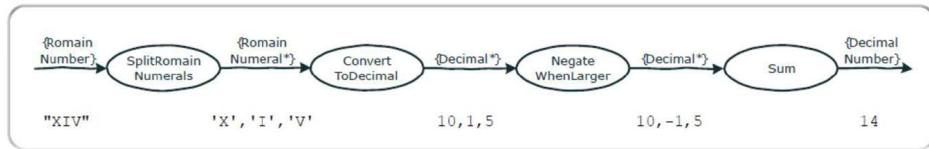


Abbildung 1: Beispiel Mutual Oblivion

- Integration Operation Segregation Principle

Das „Integration Operation Segregation Principle“ besagt, dass man Logik Operationen („Operation“) und deren Verknüpfung und die Integration in das Programm („Integration“) strikt trennen sollte. Durch die Vermischung von Logik und Verknüpfung wird ein System schwieriger zu lesen und schnell unübersichtlich. Es steigt die Anfälligkeit für Fehler, wodurch die Maintainability erschwert wird.

Genau dem wird mit dem „Integration Operation Segregation Principle“ entgegengewirkt.

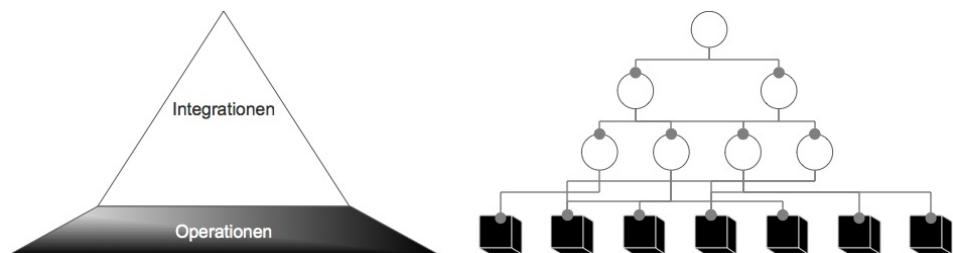


Abbildung 2: Integration Operation Segregation

Es sollen also möglichst viele kleine, voneinander unabhängige Module oder Funktionen entworfen werden, die nur wissen, welche Daten sie erhalten und welche sie zurückgeben. In Integrationsfunktionen, welche nur den Datenfluss der anderen Funktionen kennen, wird der Datenfluss durch die kleinen Module dann gesteuert und richtig gelenkt.

1.1.2 Diagramme

In seinem Buch „The Architect’s Napkin“, in dem Ralf Westphal die Grundzüge von Flow Design erläutert, zeigt er auch auf, wie man seine Vorstellungen von Software-Entwurf in Form von Diagrammen visualisieren kann. Diese Diagramme sind auf verschiedenen Ebenen aufgeteilt.

Angefangen von dem System als Ganzes (mit den Aktoren und zugehörigen Ressourcen), wird immer weiter ins Detail gegangen. Das ganze System wird möglicherweise in verschiedene Teile getrennt, welche über Interfaces miteinander kommunizieren. Innerhalb dieser Teile können logische Funktionen aufgerufen werden oder das System wird weiter in Teil-systeme, entsprechend der Aufgaben, getrennt. Dies wird so lange weiter gedacht, bis alle Teile und Aufgaben des Systems abgedeckt sind.

- System-Umwelt-Diagramm

Das System-Umwelt-Diagramm stellt den Startpunkt für die Architektur eines neuen Softwaresystems dar. Bei dieser bewusst einfach gehaltenen Diagrammart geht es um die Betrachtung des Großen und Ganzen, um bei der Systemanalyse den Überblick zu behalten. Die Umwelt ist aufgeteilt in Akteure, die das System nutzen und Ressourcen, auf die das System zugreifen kann. Akteure sind vom System abhängig, das System selbst hängt wiederum von seinen Ressourcen ab. Auch andere Softwaresysteme können als Akteure oder Ressourcen fungieren. [3]

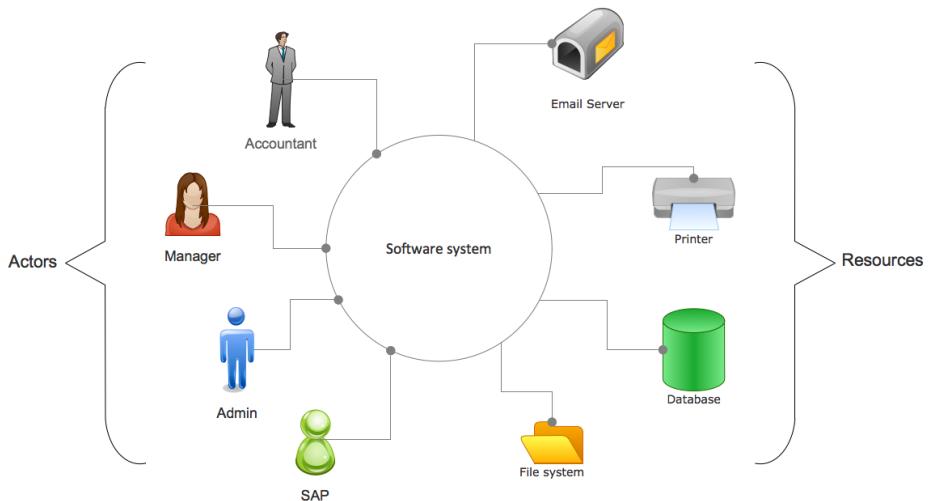


Abbildung 3: Beispiel System-Umwelt-Diagramm

- Maskenprototyp

Ein Maskenprototyp hilft dabei ein System besser zu verstehen und verschiedene Programmabläufe und Aktionen darzustellen. Auf einer Abbildung eines Software Prototyps können Interaktionen wie z.B. Delegationen gekennzeichnet werden.

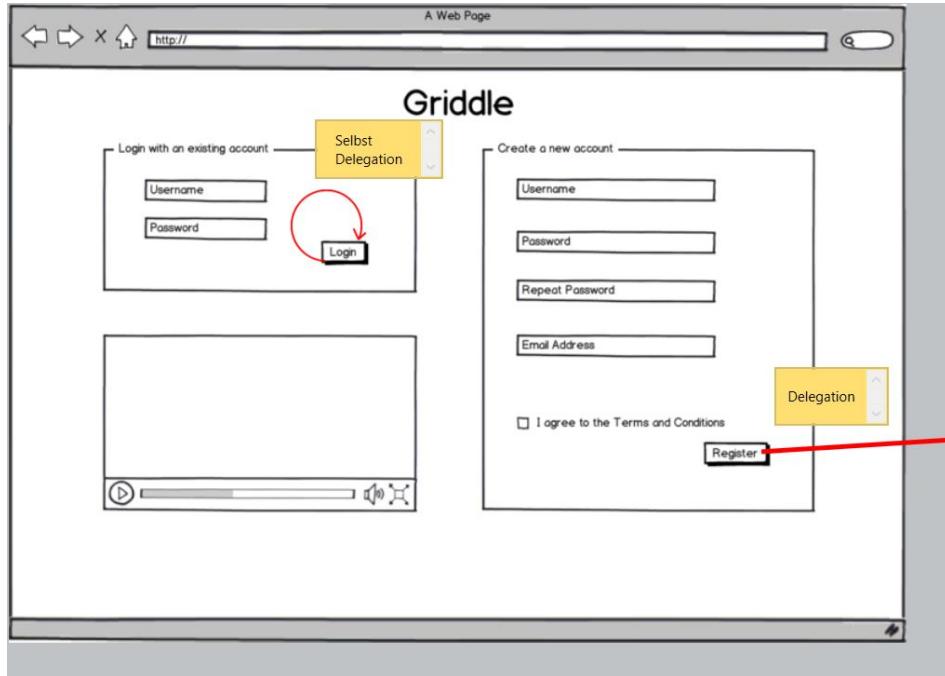


Abbildung 4: Beispiel Maskenprototyp

- Flow-Diagramm

In einem Flow-Diagramm liegt der Fokus auf dem Datenfluss und nicht auf dem Kontrollfluss. Es ist nicht so wichtig, wo die Kontrolle zu einem bestimmten Zeitpunkt ist. Vielmehr ist entscheidend, wann wo welche Daten wohin fließen. Flow-Diagramme fordern einen Umstieg vom Kontrollflussdenken hin zum Datenflussdenken.[2]

Jede Funktionseinheit ist simpel aufgebaut mit einem Eingang und einem Ausgang. Die Bezeichnung einer solchen Funktionseinheit ergibt sich aus ihrem Verhalten. Während sich die übliche Objektorientierung eher an Substantiven entlang hangelt, reichen für Flow Design Funktionseinheiten meist kurze Verben aus, um die Aktion zu beschreiben, die auf den Daten ausführt wird. Die bei der Ausführung fließenden Daten werden einfach über die Pfeile geschrieben. Wichtig im Vergleich mit UML ist bei diesem Diagrammtyp, dass die Datenflusspfeile keine Abhängigkeiten zwischen den einzelnen Einheiten beschreiben.[2]

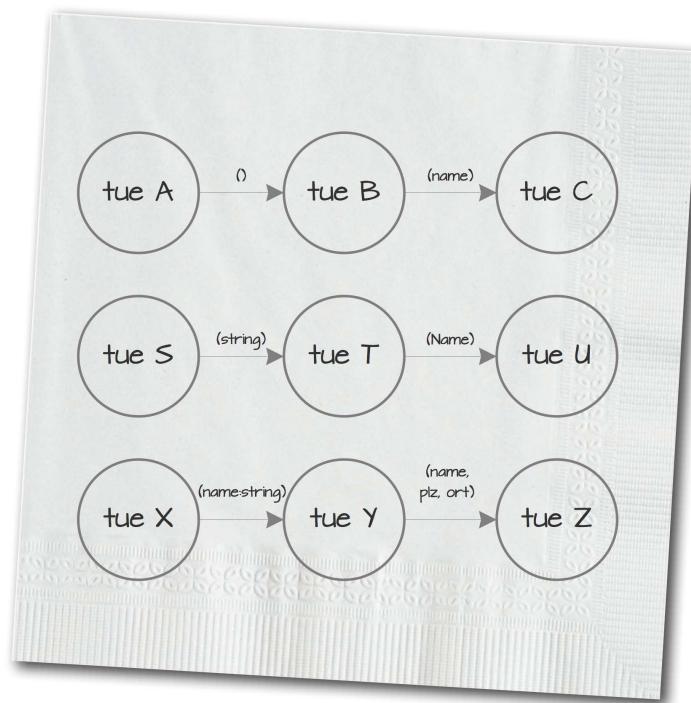


Abbildung 5: Beispiel Flow Diagramm

2 Spezifikation

2.1 Einleitung

2.1.1 Zweck der Spezifikation

In der Spezifikation sind alle Anforderungen an die zu entwickelnde Software festgelegt. Somit dient sie als Grundlage für alle weiteren entstehenden Dokumente während der Entwicklung. Sie bildet den roten Faden für die Entwicklung und sollte von allen Team-Mitgliedern berücksichtigt werden. Das Team verpflichtet sich außerdem die Spezifikation aktuell zu halten und mit anderen Dokumenten, vor allem Entwurf und Implementierung abzustimmen.

2.1.2 Leserkreis der Spezifikation

Folgende Personengruppen gehören zum Leserkreis der Spezifikation:

- Das Entwicklerteam
 - Daniel Glinka - Projektmanager
 - Daniel Kratzel - Chefdesigner
 - Johannes Schlier - Qualitätsmanager
 - Johannes Steinhülb - Entwicklungsleiter
- Der Betreuer
 - Prof. Dr.-Ing. Reinhard Schmidt
- Der Kunde
 - Kevin Erath (IT-Designers GmbH)

2.1.3 Einsatzbereich und Ziele von Flow Design

Die zu entwickelnde Software soll als Entwurfstool innerhalb der Firma des Kunden IT-Designers GmbH verwendet werden können. Die Software kann durch Diagramme und Schaubilder den Entwurf und die Architektur eines Systems darstellen und bei dem weiteren Entwurf und der Implementierung helfen.

2.1.4 Verwendete Abkürzungen und Schreibweisen

- **WPF** - Windows Presentation Foundation
- **MVVM** - Model View View-Model
- **UI** - User Interface
- **GUI** - Graphical User Interface

2.2 Allgemeine Beschreibung

2.2.1 Einbettung

Flow Design ist eine Desktopanwendung, die unter Windows 10 ausgeführt wird. Auf den ausführenden Computern sollte das .NET-Framework 4.6 installiert sein. Für die Windows Presentation Foundation muss DirectX 9 auf dem Zielrechner installiert sein.

2.2.2 Überblick über die Funktionen

Flow Design ist eine Desktopanwendung, welche

- einen Einzelnutzerbetrieb unterstützt.
- die Erstellung von Flow Design Diagrammen ermöglicht.
- die Erstellung von Flow Design Projekten ermöglicht.

2.2.3 Mengengerüst

- Es arbeitet immer nur ein Nutzer an einem Projekt.
- Es gibt keine Begrenzung bei der Erstellung von Flow Design Projekten in der Software.
- Es gibt keine Begrenzung für Flow Design Diagrammen in einem Flow Design Projekt durch die Software.

2.2.4 Profil der Flow Design Nutzer

Die Benutzer der Software arbeiten in der Softwareentwicklung und haben sehr gute Computerkenntnisse.

2.2.5 Entwicklungswerkzeuge

Bei der Entwicklung von Flow Design werden folgende Werkzeuge verwendet:

- Visual Studio 2015 - Entwicklungsumgebung
- ShareLaTeX - Erstellung von Dokumenten
- Lucidchart - Erstellung von Diagrammen (UML, Use-Cases...)
- Git - Versionverwaltung
- Bitbucket - Versionsverwaltungsseite / Issue Tracking
- Adobe Illustrator - Icons
- Trello - Kanban Board

2.3 Einzelanforderung

2.3.1 Nicht-Funktionale Anforderungen

- Aussehen und Handhabung
 - Die Benutzerschnittstelle ist übersichtlich gegliedert und intuitiv zu bedienen.
- Benutzbarkeit
 - Das Programm ist auch von einem Laien ohne Einweisung bedienbar. Die Dokumentation beschreibt die gesamte Funktionalität der Software.
- Wartbarkeit/Änderbarkeit
 - Während der Entwicklung wird auf sorgsame Dokumentation sowohl im Code als auch in externen Dokumenten geachtet. Die Entwickler halten die Dokumente immer aktuell und konsistent. So soll einem Softwareentwickler die Wartung und Erweiterung von Flow Design erleichtert werden.
 - Die Entwickler verpflichten sich, die **C# Coding Conventions** zu berücksichtigen, um die Strukturiertheit und Lesbarkeit des Codes sicherzustellen.
- Sicherheitsanforderungen
 - Macht der Benutzer falsche Angaben, so werden Fehlermeldungen angezeigt und bestimmte Funktionen bleiben gesperrt. Bei kritischen Aktionen wie Löschen und Abbrechen eines Projekts ist die Bestätigung des Nutzers erforderlich.
- Ausfallsicherheit
 - Bei fehlerhaften Eingaben fängt Flow Design diese ab, gibt die betreffenden Fehlermeldungen aus und fordert den Nutzer zur erneuten Eingabe auf. Auch bei sonstigen Fehlern wird immer eine Fehlermeldung ausgegeben und in der Log-Datei vermerkt.
- Skalierbarkeit
 - Flow Design sollte Auflösungen von 1024 x 768 bis 1920 x 1200 Pixeln unterstützen, d.h. die Desktop-Anwendung sollte bei diesen Einstellungen immer noch korrekt angezeigt werden.

2.3.2 Funktionale Anforderungen

- Diagramm 1 - System Umwelt Diagramm
 - Der Benutzer hat die Möglichkeit, Systeme in Form von Kreisen darzustellen.
 - Es können Compiler und andere Ressourcen in Form von einer „Tonne“ erzeugt werden.
 - Zwischen den einzelnen Bausteinen können Beziehungen erstellt werden.
 - Es gibt die Möglichkeit, jeden Baustein oder seine Beziehung mit einem Kommentar zu versehen.
- Diagramm 2 - Maskenprototyp
 - Der Benutzer hat die Möglichkeit, Bilder und Prototypen in das Diagramm zu importieren.
 - Interaktionen können auf dem importierten Bild durch Pfeile gekennzeichnet werden.
 - * Selbstdelegation
 - * Interaktion mit Response
 - * Interaktion ohne Response
- Diagramm 3 - Flow Design
 - Stellt die Interaktionen des Maskenprototyp in Form von Flow Design dar.
 - Input und Output der einzelnen Operationen wird mit Pfeilen dargestellt. Es ist möglich, die Datentypen von Input und Output darzustellen.
 - Eine Operation kann wieder ein Flow Design Diagramm enthalten.
 - Datenflüssen kann ein Kommentar angehängt werden.

2.4 Anwendungsfälle

2.4.1 Anwendungsfalldiagramm

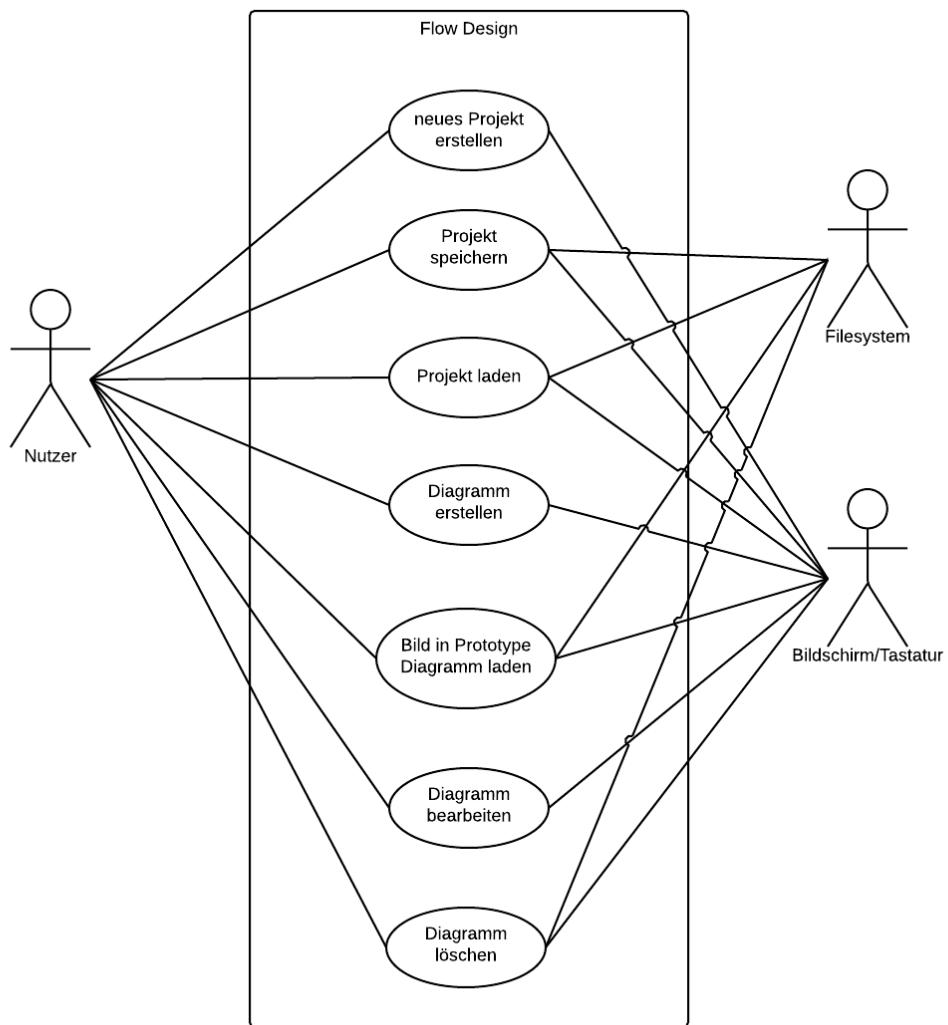


Abbildung 6: Anwendungsfalldiagramm

2.4.2 Usability Patterns

- Systemstatus - Der Status der Anwendung wird leicht erkennbar in einer Statusleiste am unteren Rand des Fensters angezeigt.
- Automatisches Speichern - Die Anwendung wird nach einer Änderung automatisch

gespeichert um einen Datenverlust zu vermeiden.

2.4.3 Kontextdiagramm

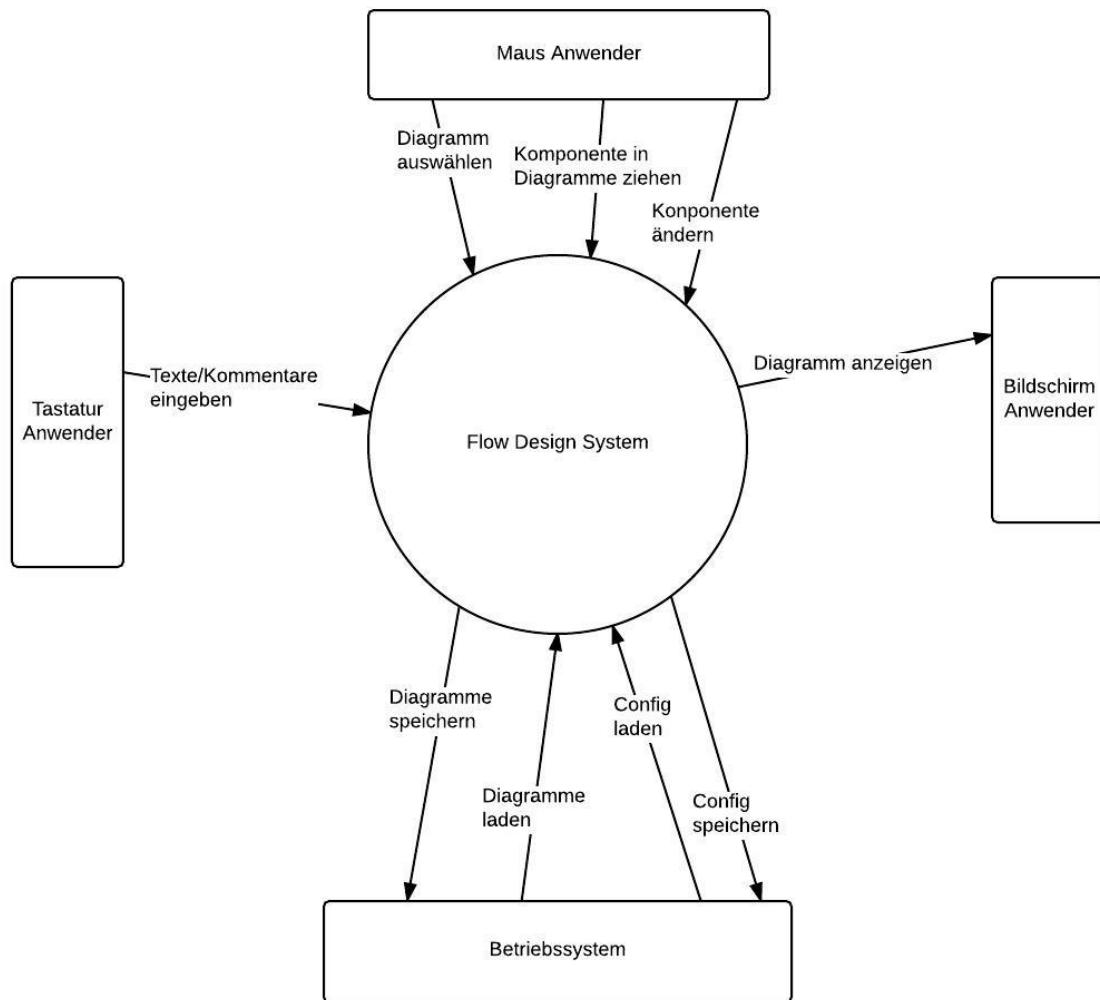


Abbildung 7: Kontextdiagramm

2.5 Anhang

2.5.1 Begriffslexikon

Begriff	Model-View-ViewModel
Bedeutung	Model-View-ViewModel ist ein Entwurfsmuster um die Logik von der Darstellung zu trennen.
Begriff	Config
Bedeutung	Konfigurationsdatei.

3 Projektplan

3.1 Einleitung

3.1.1 Zweck des Projektplans

Dieser Projektplan dokumentiert die Ziele des Projekts, die Zeitplanung für deren Verwirklichung und alle relevanten Bedingungen für das Projekt. Da der Projektablauf nicht genau vorausgesagt werden kann, muss dieser Projektplan mit dem tatsächlichen Projektablauf synchron gehalten und damit verändert werden.

3.1.2 Projektüberblick

In diesem Projekt soll eine Software entstehen um Flow Design Diagramme zu modellieren (siehe Spezifikation Seite 6).

3.1.3 Entwicklungsphilosophie

Bei der Bearbeitung des Projektes sollten ingenieurmäßige Prinzipien zur Anwendung kommen, um eine hohe Qualität der Ergebnisse und die Einhaltung der Ablaufplanung sicherzustellen. Daher ist auf Einhaltung aller gegebenen Normen und Standards zu achten. Alle Dokumente werden immer aktuell gehalten, um am Projektende eine ausführliche Dokumentation des Projektes und des Projektlaufes zu haben. Dies gewährleistet eine Funktionalität des fertigen Softwareprodukts.

3.1.4 Vertragliche Grundlagen

Die Teammitglieder verpflichten sich ohne eine feste Anstellung die Software zu entwickeln. Das Entwicklerteam übernimmt keinerlei Haftung für die Schäden, die durch Fehlfunktionen der entwickelten Software (Flow Design) verursacht werden. Das für die Entwicklung gewählte Vorgehensmodell, ist das Iterative Modell. Um sicherzustellen, dass das Projekt im Zeitplan realisiert wird und die einzelnen Module ausführlich getestet werden, wird ein Kanban Board verwendet.

3.2 Risiken

3.2.1 Risiken, ihre Bewertung und Gegenmaßnahmen

- Personeller Ausfall

Mittlere Eintrittswahrscheinlichkeit. Sollte das ganze Entwicklerteam ausfallen, was eine Eintrittswahrscheinlichkeit von nahe zu null hat, würde dies zu einem Abbruch des Projekts führen.

Durch regelmäßige Treffen und Nachrichtenkontakt informiert jeder Projektteilnehmer die anderen über seine Arbeit und seinen Fortschritt. Dadurch haben alle Entwickler einen Überblick über den Stand der anderen und können auf kurzzeitige Ausfälle durch eine Umverteilung reagieren. Zusätzlich müssen alle beteiligten Entwickler sehr ausführlich ihren Code, sowie die restlichen Produktbestandteile kommentieren und erläutern. Sollte das Entwicklerteam komplett ausfallen, ist damit zu rechnen, dass das Projekt eingestellt wird. In allen Fällen muss der Betreuer informiert werden.

- Terminprobleme

Mittlere Eintrittswahrscheinlichkeit. Um Terminproblemen entgegen zu wirken, wird ein Terminplan erstellt und aktuell gehalten. Außerdem werden den Teammitgliedern verschiedene Zuständigkeitsbereiche zugeteilt. Ein „gold plating“, also die Perfektionierung von kleinen, eher unbedeutenden Bestandteilen, soll vermieden werden.

- Entwicklung der falschen Funktionalität

Geringe Eintrittswahrscheinlichkeit. Um einer falschen Funktionalität entgegen zu wirken, wird dem Kunden regelmäßig eine Version der Software geliefert, welche von diesem geprüft wird und er den Entwicklern Rückmeldung gibt. Sollte eine Änderung vorliegen, ist diese sofort in die Spezifikation zu übernehmen.

- Kommunikationsprobleme im Team

Sehr geringe Eintrittswahrscheinlichkeit. Missverständnisse und Kommunikationsprobleme werden durch regelmäßige persönliche Treffen, sowie häufigen Nachrichtenkontakt vermieden. Die durch Missverständnisse entstandene Zeitverschiebung muss rechtzeitig im Terminplan vermerkt werden.

3.3 Entwicklungsplan

3.3.1 Zeitplan und Meilensteine

- Kick-off - Woche 1
- Treffen IT-Designers - Woche 1
- Überarbeitung Projektplan - Woche 1-2
- UI-Prototyp - Woche 2
- Abnahme Entwurf - Woche 2-3
- Implementierung - Woche 3-7
- Abgabe Implementierung - Woche 7-8
- Abnahme IT-Designers - Woche 8
- Abschlusspräsentation - Woche 9
- Repository Übergabe - Woche 9

3.4 Entwicklungsprozess

3.4.1 Dokumentation

Folgende Dokumente sollen während des Projekts erstellt und gepflegt werden:

- Projektplan
- Spezifikation

3.4.2 Qualitätssicherung

Während des gesamten Entwicklungsprozesses werden Unit-/Integrations und Systemtest durchgeführt. Zudem muss der Projektfortschritt regelmäßig erfasst und die Zeitplanung

ggf. daran angepasst werden. Damit sollen unerwartete Terminschwierigkeiten verhindert werden.

3.5 Projektorganisation

Projektmanager	Daniel Glinka MatrikNr: 749512 Email: daglit01@hs-esslingen.de
Entwicklungsleiter	Johannes Steinhülb MatrikNr: 749818 Email: jostit01@hs-esslingen.de
Usability und GUI-Design	Daniel Kratzel MatrikNr: 749318 Email: dakrit03@hs-esslingen.de
Qualitätsmanager	Johannes Schlier MatrikNr: 749815 Email: joscit13@hs-esslingen.de
Betreuer	Prof. Dr.-Ing. Reinhard Schmidt Email: reinhardt.schmidt@hs-esslingen.de
Kunde	Kevin Erath Email: erath@it-designers.de

4 Benutzeroberfläche

4.1 Einleitung

Im Folgenden werden Entwürfe für die grafische Oberfläche der zu entwickelnden Software dargestellt. Dabei werden folgende Entwicklungsstadien gezeigt:

Erste Ideen

Konkrete Entwürfe

Grafischer Prototyp

Finale Oberfläche

4.1.1 Erste Ideen

Die nachfolgenden Abbildungen zeigen die ersten Überlegungen zur Benutzeroberfläche. Zu Beginn konzentrierten wir uns auf die Ansichten der einzelnen Diagrammarten und beschäftigten uns parallel mit der möglichen Navigation zur Bearbeitung dieser Diagramme.

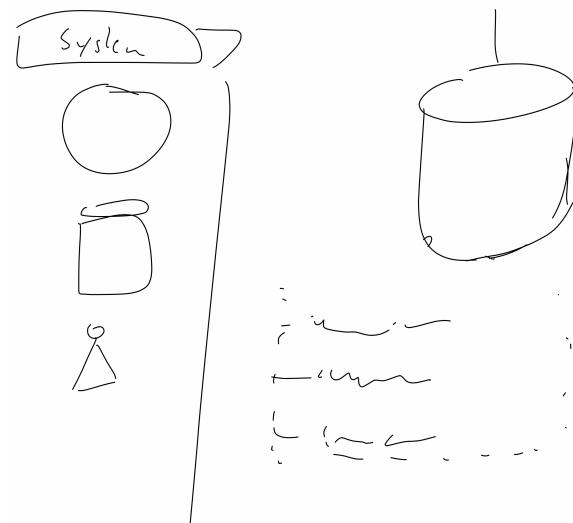


Abbildung 8: System-Umwelt Diagramm Ressourcen

4 Benutzeroberfläche

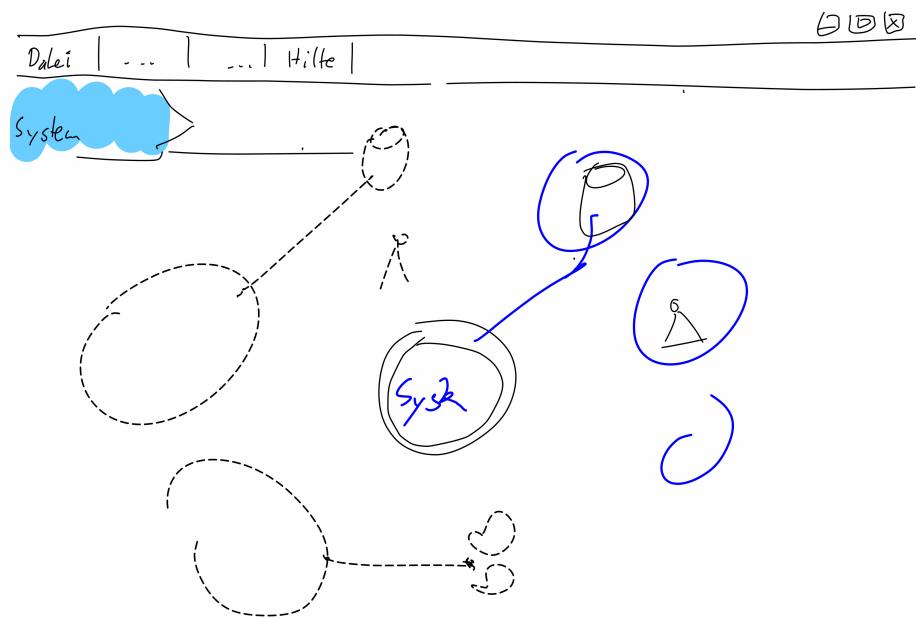


Abbildung 9: System-Umwelt Diagramm

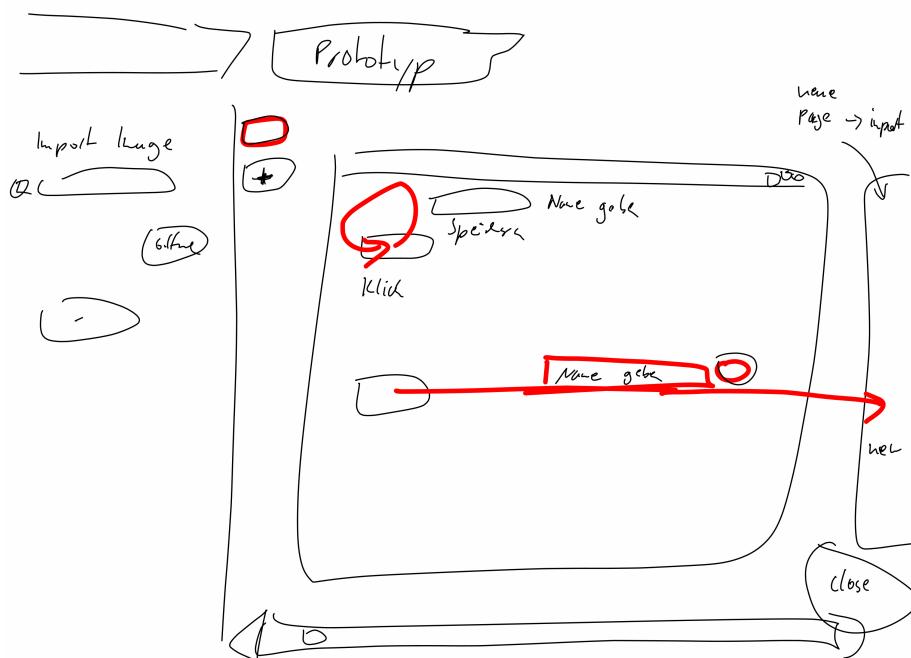


Abbildung 10: Maskenprototyp

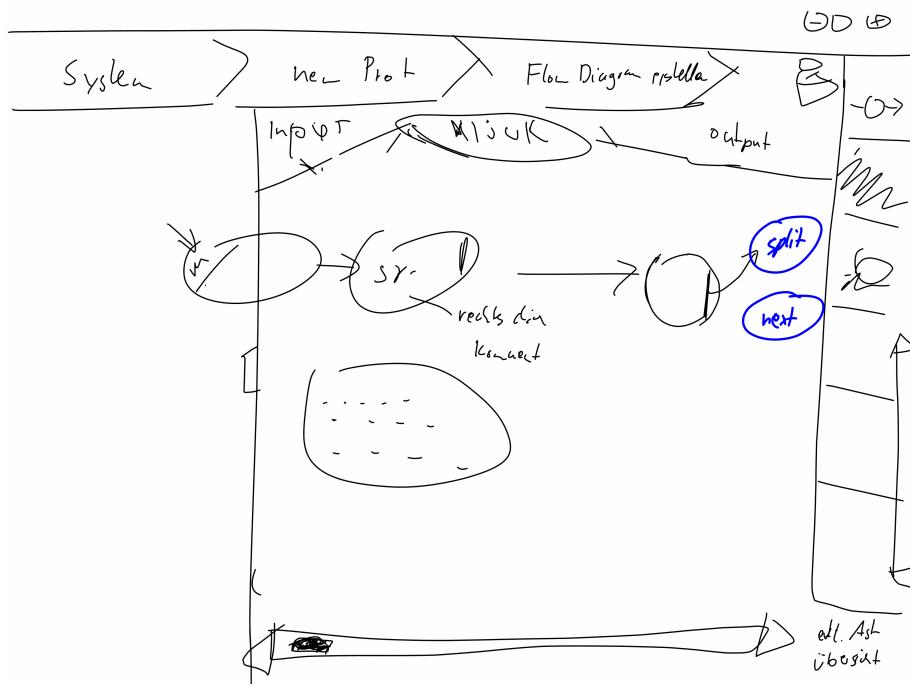


Abbildung 11: Flow Design

4.1.2 Konkrete Entwürfe

Nach der kreativen Phase mit vielen verschiedenen Ideen und dem Abwägen von Vor- und Nachteilen, erstellten wir erste konkrete Entwürfe. Bei der Erarbeitung eines einheitlichen Konzepts der Benutzeroberfläche orientierten wir uns an verschiedenen Gestaltungskriterien und Usability Prinzipien.

4 Benutzeroberfläche

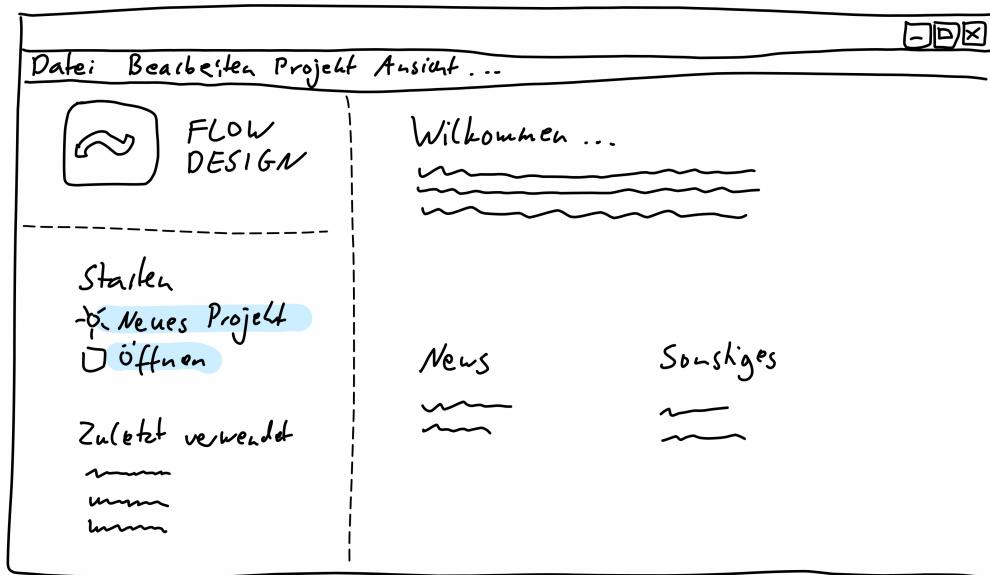


Abbildung 12: Startseite

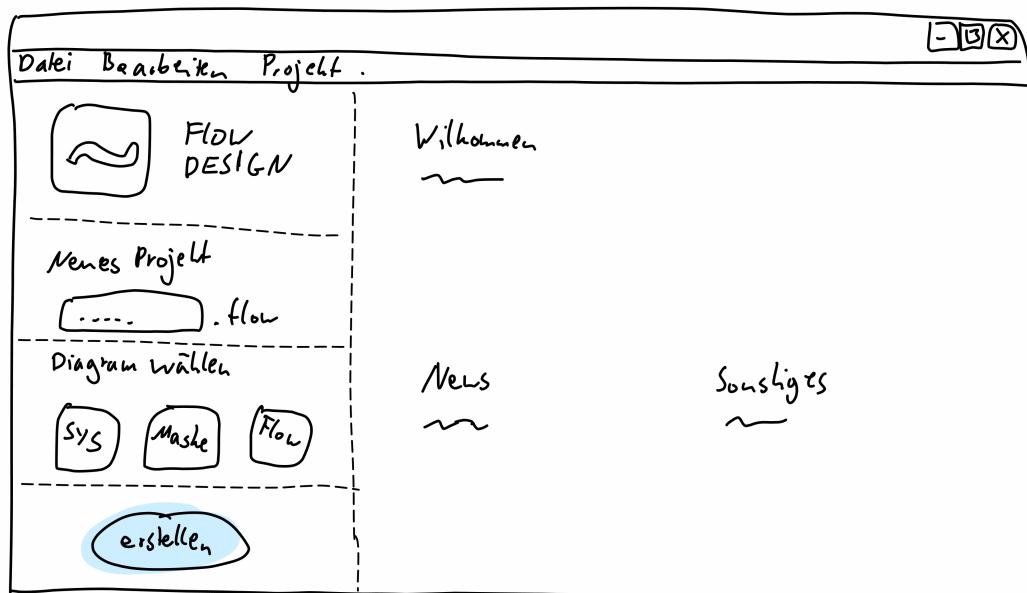


Abbildung 13: Projekt erstellen (Option 1)

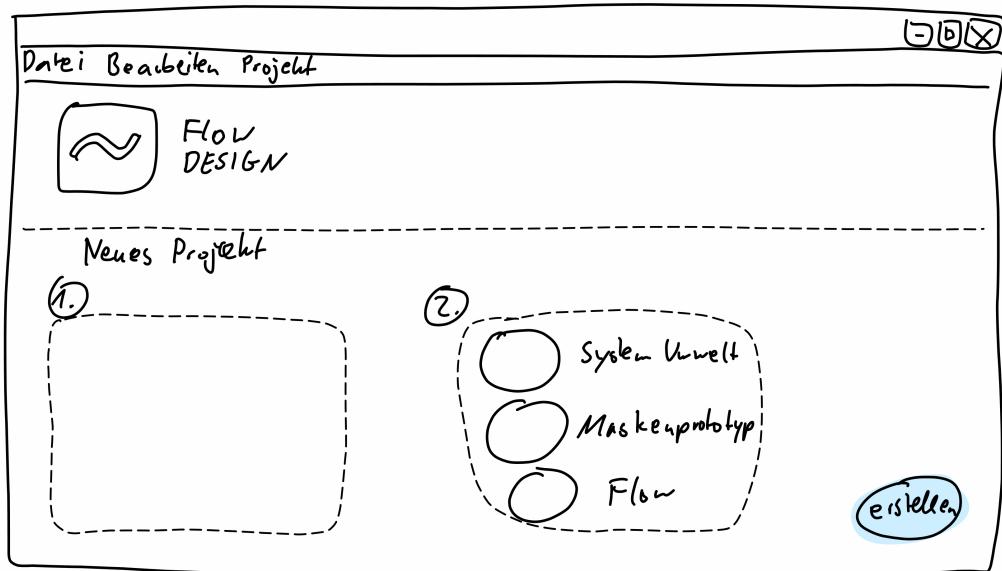


Abbildung 14: Projekt erstellen (Option 2)

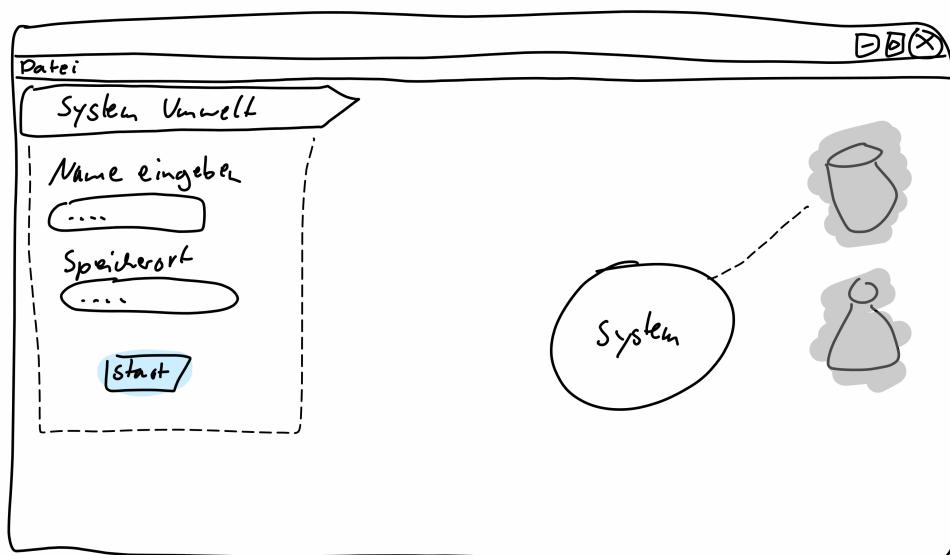


Abbildung 15: System Umwelt

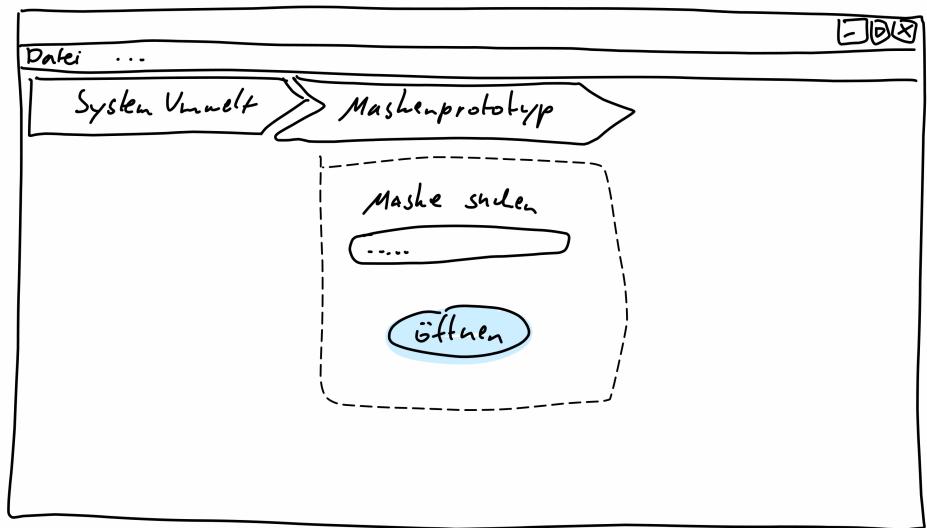


Abbildung 16: Maskenprototyp wählen

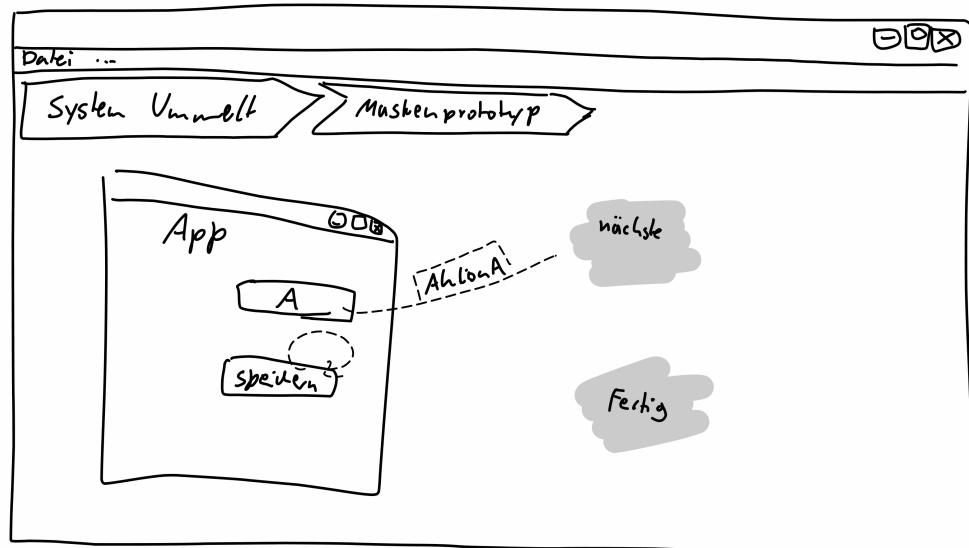


Abbildung 17: Maskenprototyp

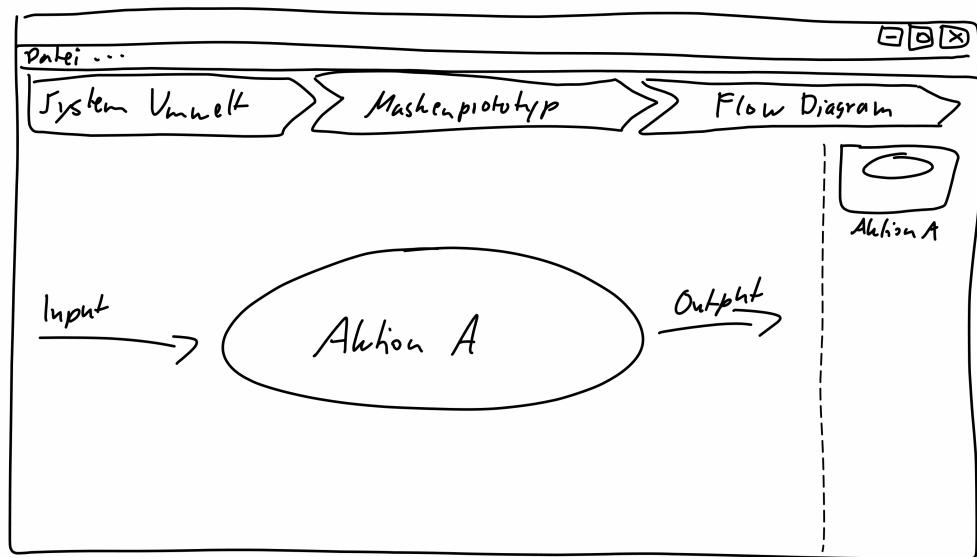


Abbildung 18: Flow Diagramm

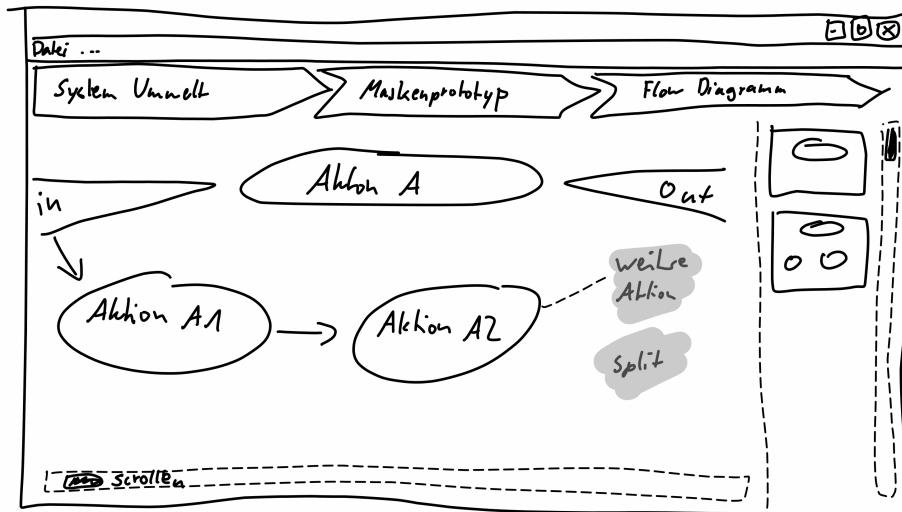


Abbildung 19: Flow Diagramm Detailansicht

4.1.3 Grafischer Prototyp

Um optisch einen besseren Eindruck über die spätere Software zu bekommen, entwickelten wir in Photoshop einen grafischen Prototyp auf Basis der bisherigen Entwürfe.

Die Startseite beinhaltet einen Willkommensfenster mit einer kurzen Beschreibung zu Flow Design und den neusten Meldungen über Updates. Es besteht die Möglichkeit ein neues Projekt anzulegen oder ein bereits bestehendes Projekt zu öffnen. Zudem werden Links zu den zuletzt verwendeten Projekten bereitgestellt, über welche diese geöffnet werden können.

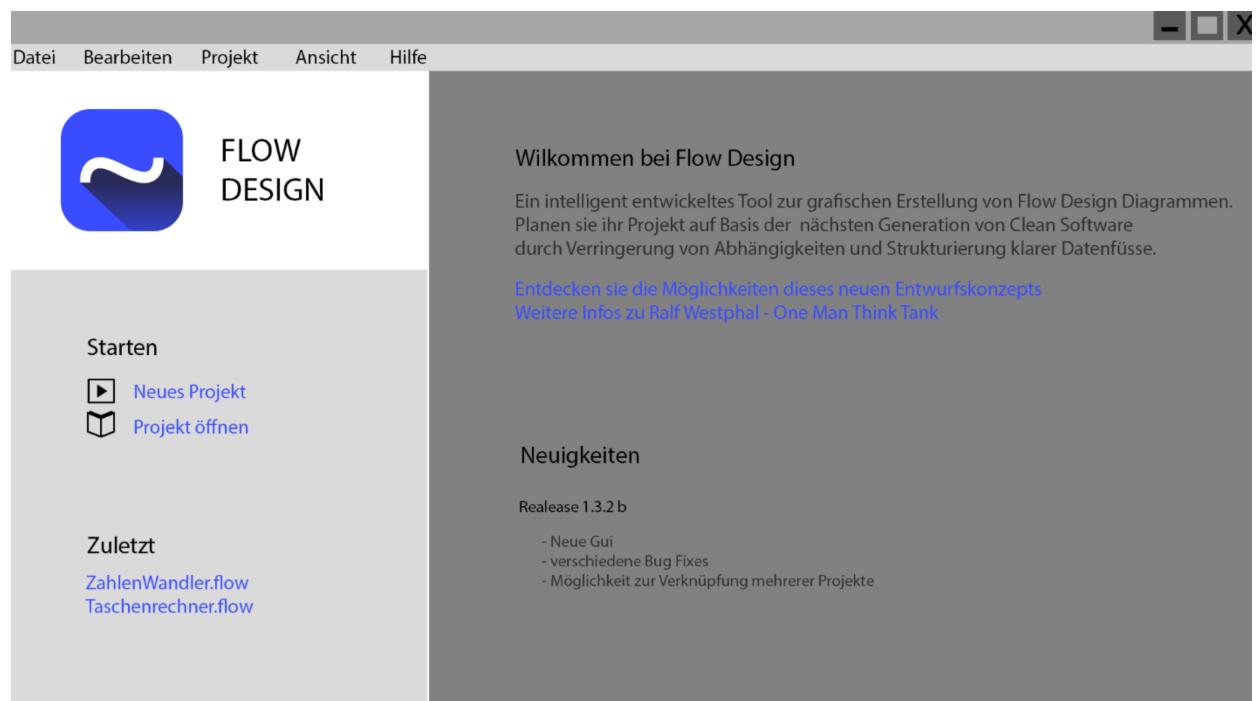


Abbildung 20: Prototyp - Startseite

Die System Umwelt sollte zu Beginn jeder Anforderungsanalyse stehen, um sich einen Überblick über das Gesamtsystem zu verschaffen ohne ins Detail zu gehen. Das System Umwelt Diagramm stellt den Einstiegspunkt eines neuen Projekts dar, welches durch die Angabe eines Namens und der Vergabe eines Speicherorts erstellt werden.

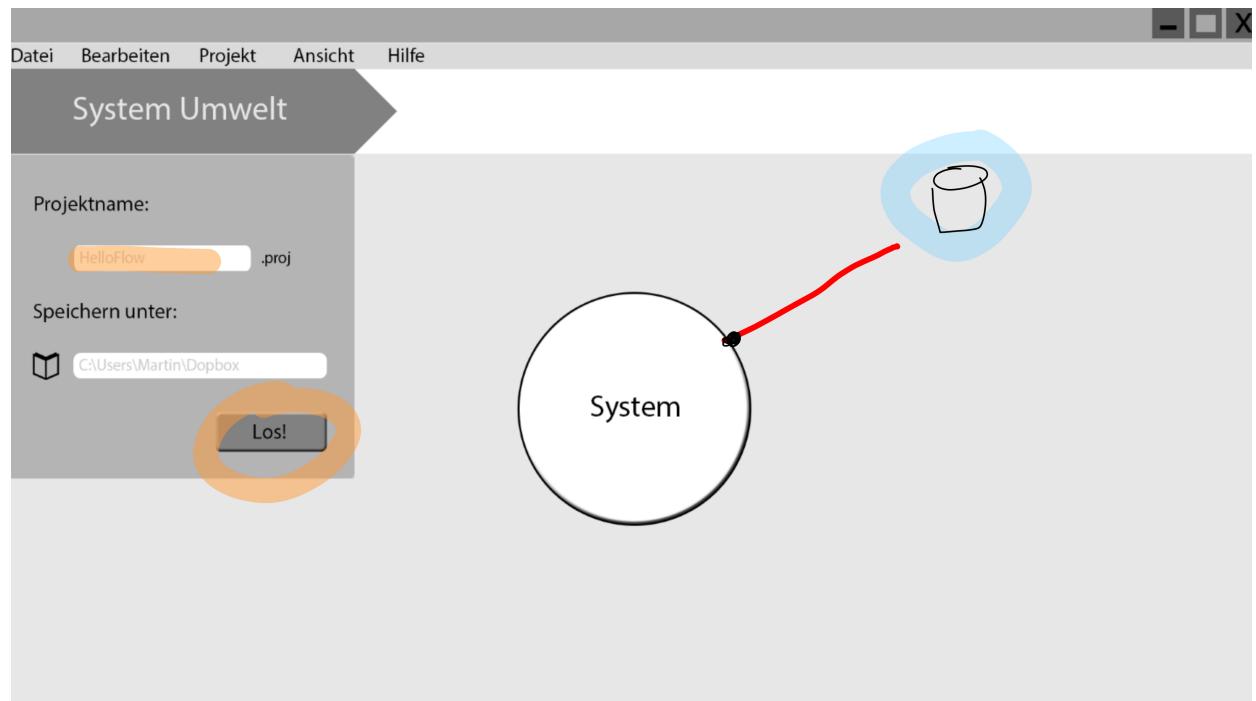


Abbildung 21: Prototyp - Projekt erstellen

Das System wird als einfacher Kreis dargestellt, an den verschiedene Komponenten angehängt werden können. Diese unterscheiden sich in Aktoren, die vom System abhängen und Ressourcen von denen das System abhängt. Zu den Aktoren zählen jegliche Personen, die das System nutzen, aber auch andere Softwaresysteme die auf das System zugreifen. Ressourcen werden vom System genutzt. Dazu zählen beispielsweise Datenbanken oder Drucker.

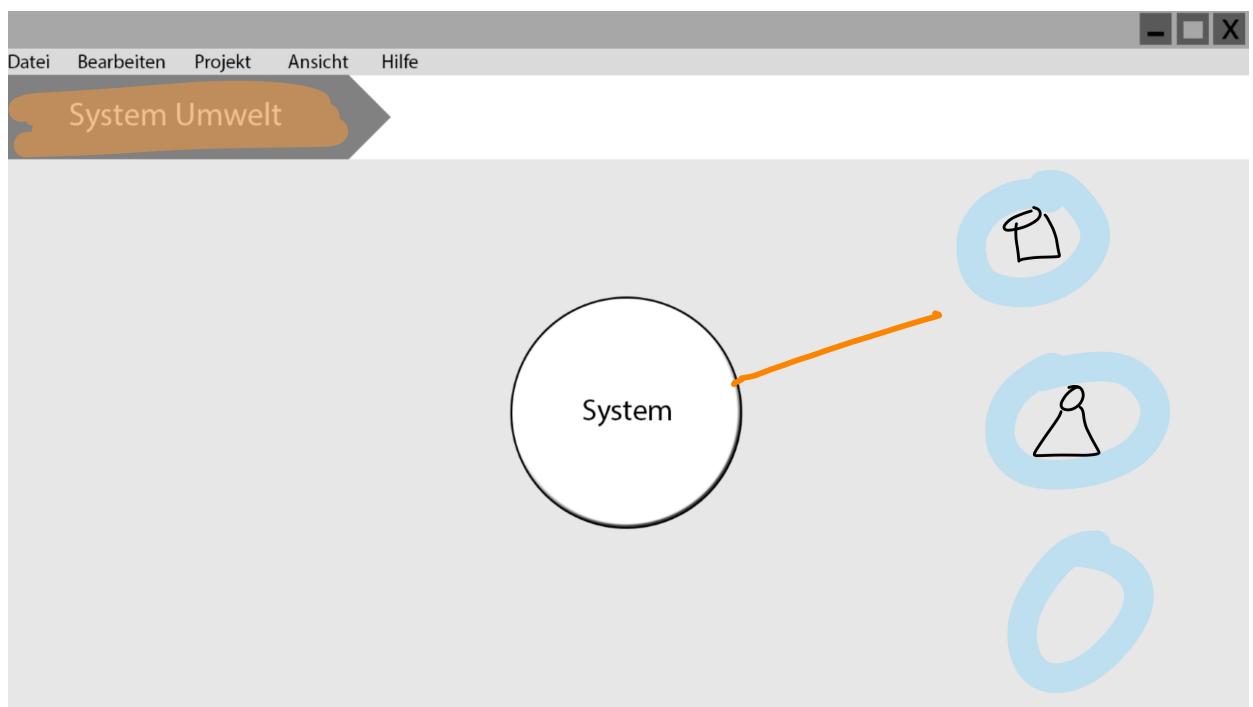


Abbildung 22: Prototyp - System Umwelt Diagramm

Die folgenden zwei Abbildungen des grafischen Prototyps zeigen das Erstellen eines Maskenprototyps durch öffnen eines Mockups und die Darstellung von Delegationen um Benutzer-Interaktionen zu veranschaulichen.

4 Benutzeroberfläche

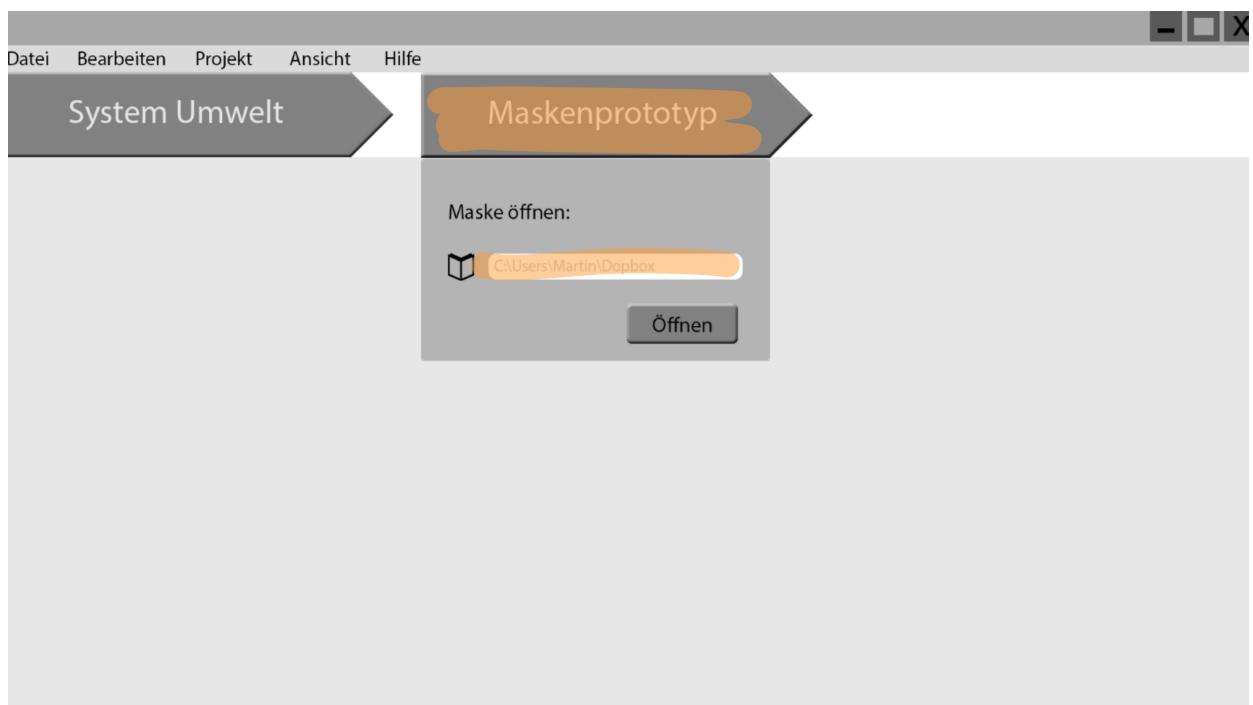


Abbildung 23: Prototyp - Maskenprototyp erstellen

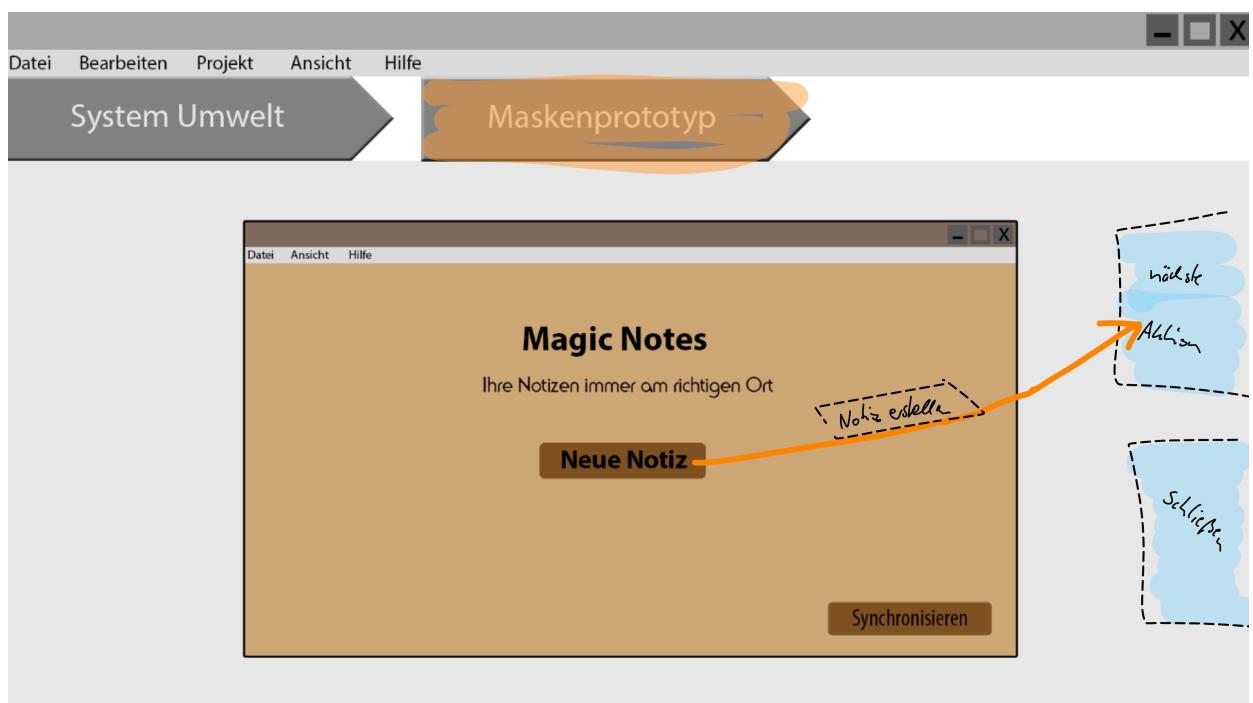


Abbildung 24: Prototyp - Maskenprototyp

Im Flow-Diagramm können die einzelnen Funktionseinheiten in Verbindung mit ihren Datenflüssen dargestellt werden. Einzelne Einheiten können in detailliertere Funktionseinheiten weiter aufgespalten werden. Sie werden auf einer neuen Flow-Ebene dargestellt.

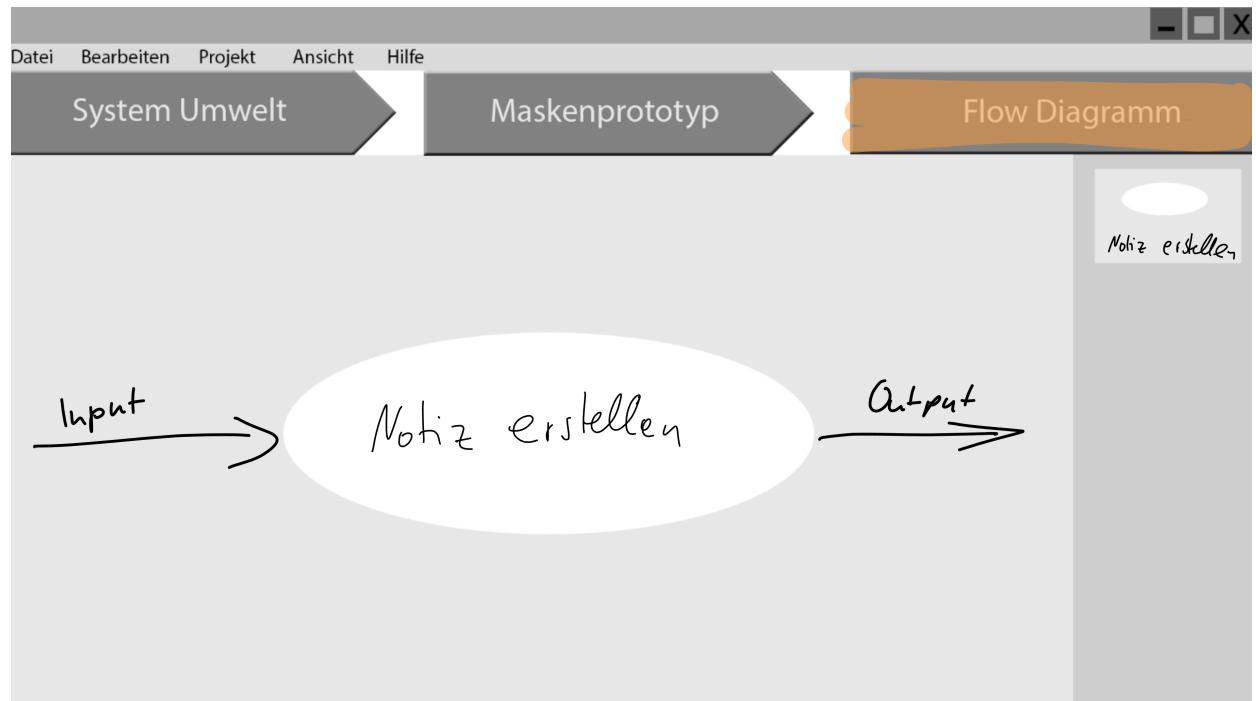


Abbildung 25: Prototyp - Flow Diagramm

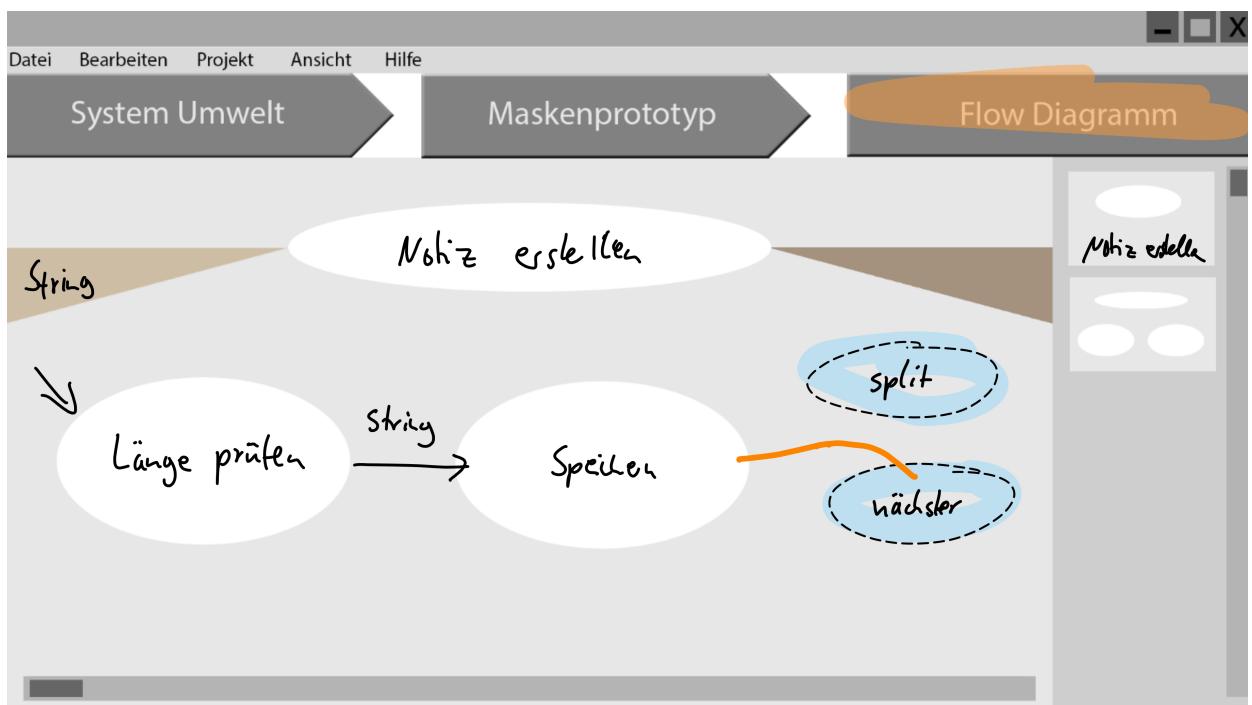


Abbildung 26: Prototyp - Flow Diagramm - nächste Ebene

4.1.4 Finale Oberfläche

In dem fertigen Programm gibt es eine Auswahlmöglichkeit um zwischen einem hellen und einem dunklen Thema zu wählen.

Im Folgenden wird jedoch nur das dunkle Thema gezeigt. Die Darstellung des hellen Themas befindet sich im Kapitel 4.1.5 auf Seite 33.

4 Benutzeroberfläche

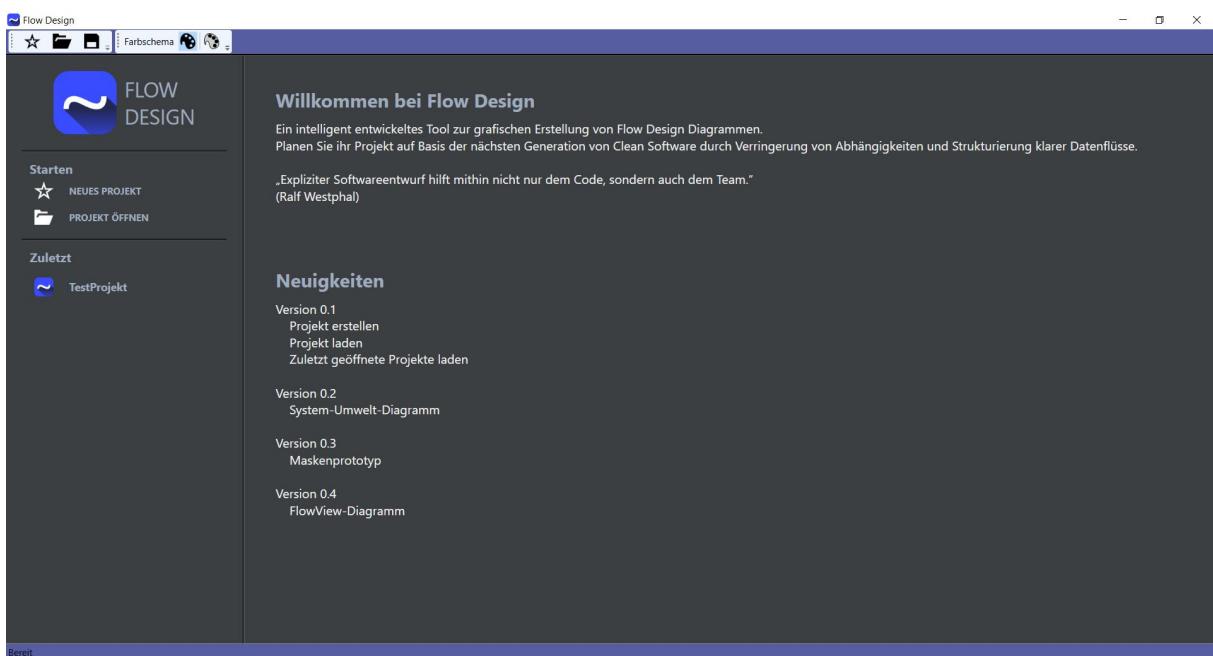


Abbildung 27: Startseite

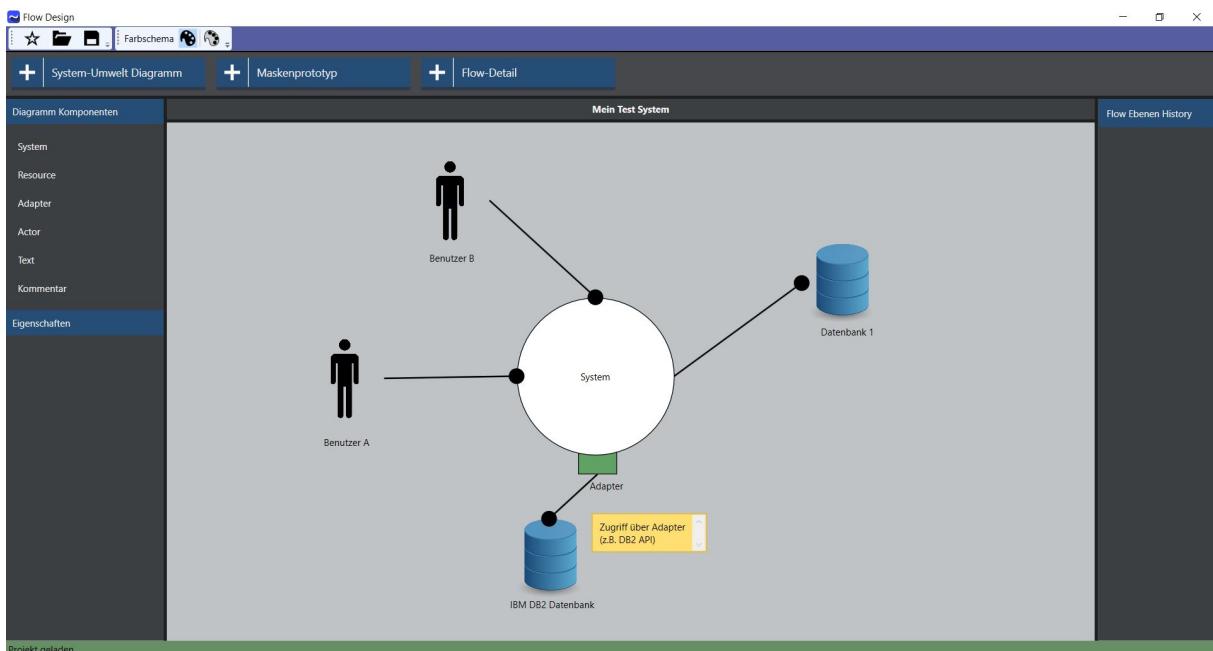


Abbildung 28: System Umwelt Diagramm

4 Benutzeroberfläche

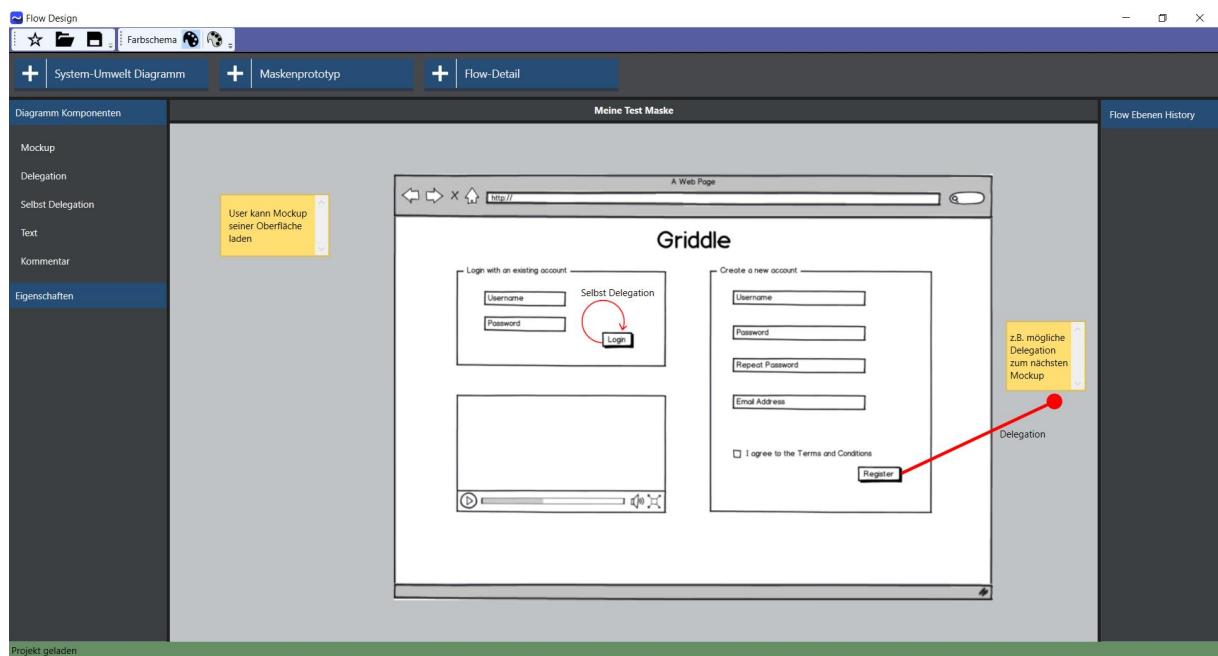


Abbildung 29: Maskenprototyp

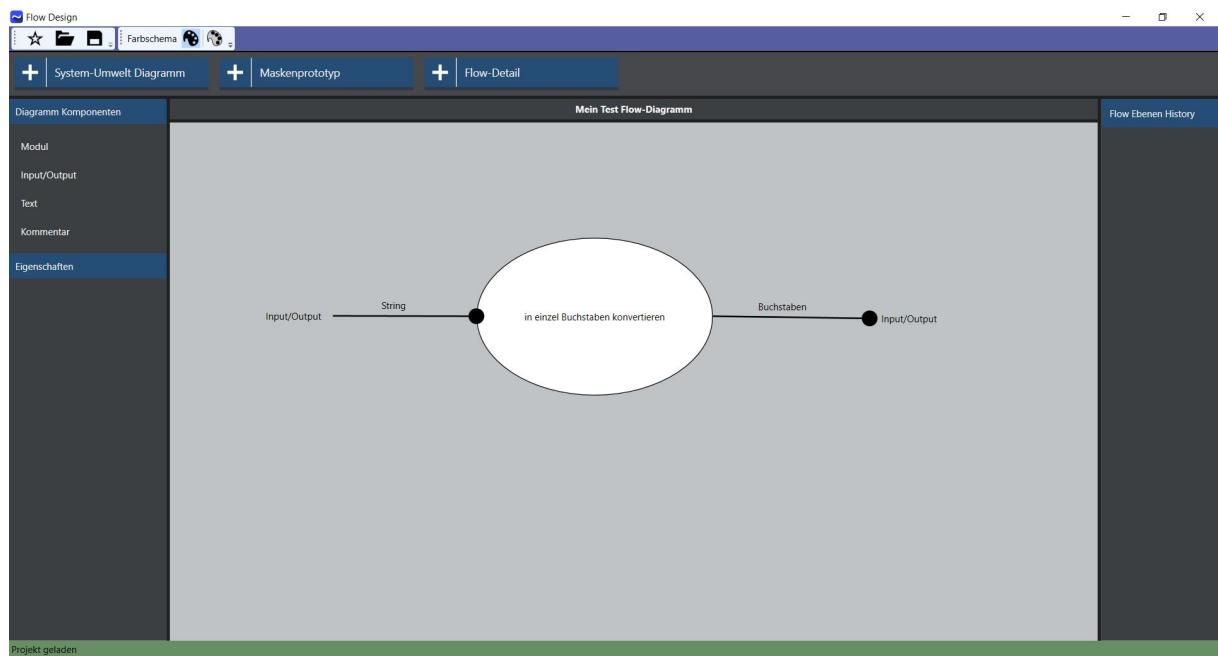


Abbildung 30: Flow Diagramm

4.1.5 Farbschema

Für die Oberfläche unseres Flow Design Prototyp, stehen dem Nutzer zwei Farbschemata zur Verfügung. Die Ansicht kann zwischen hell oder dunkel umgeschaltet werden.



Abbildung 31: Oberflächenausschnitt in hellem Farbschema

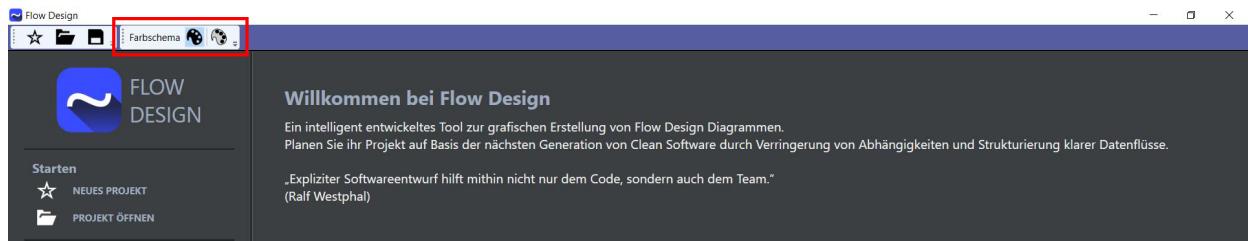


Abbildung 32: Oberflächenausschnitt in dunklem Farbschema

Zur Farbauswahl wurde das Adobe Color Tool verwendet. Es steht online kostenlos zur Verfügung und bietet Farbpaletten, die individuell oder über bestimmte Farbregeln beliebig angepasst werden können. Für die helle Ansicht haben wir uns, wie die folgende Abbildung zeigt, für eine monochromatische Farbpalette aus harmonierenden Grau- und Blautönen entschieden.

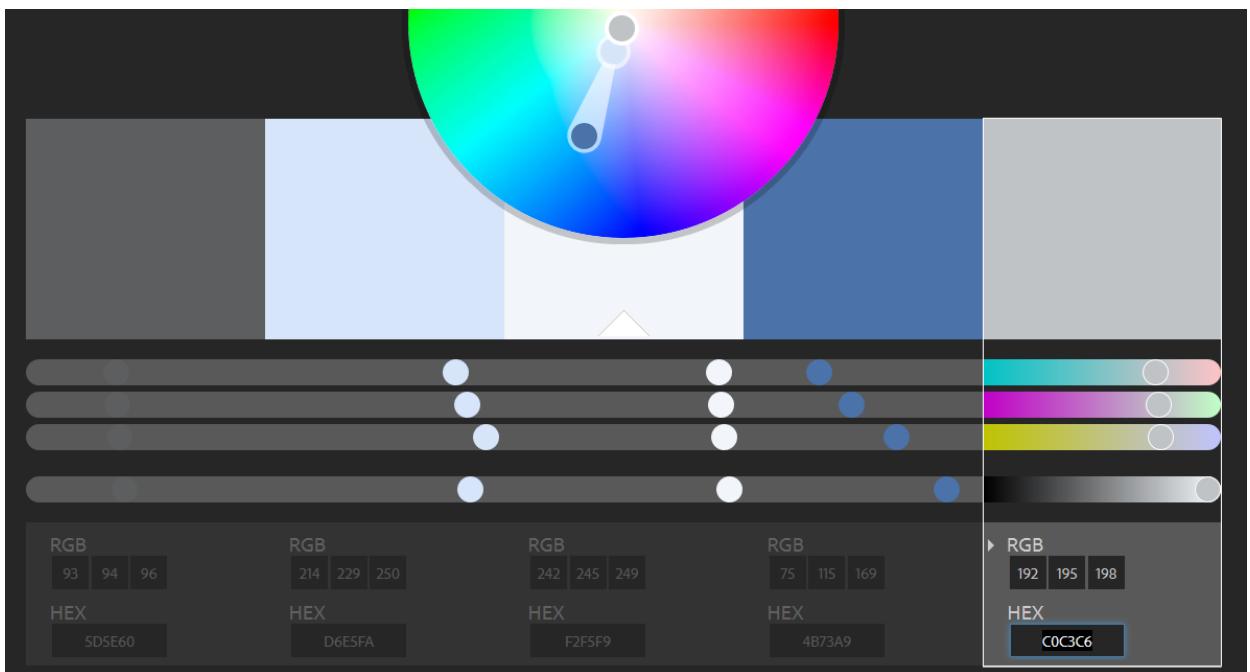


Abbildung 33: Farbpalette des hellen Farbschemas

Bei der Farbgestaltung der dunklen Version hielten wir uns an ein gedecktes Farbschema, welches von der Hauptfarbe des Logos mit dem Wert #394cff abgeleitet wurde. Der Großteil der verwendeten Farben sind, neben der Hauptfarbe, Grau- und Blautöne in verschiedenen Abstufungen.

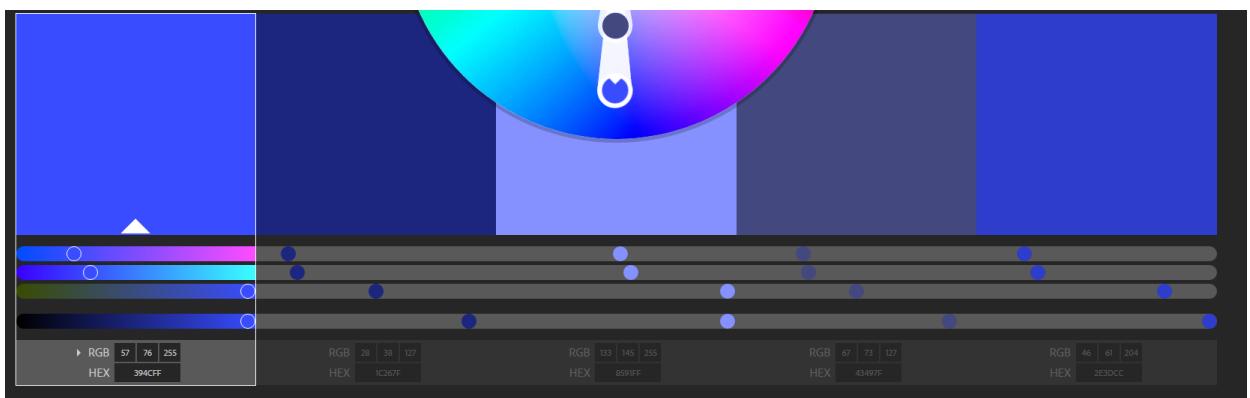


Abbildung 34: Farbpalette des dunklen Farbschemas

5 Umsetzung

5.1 Allgemein

Das Softwareprojekt im 4. Semester der Hochschule Esslingen umfasst die Bereiche Projektplanung, Problemanalyse, Spezifikation, Systementwurf, Implementierung und Test einer von einem Kunden gewünschten Software.

5.2 Projektplan

Bereits zu Beginn des Projekts legten wir mit dem Projektplan (siehe Seite 14) eine klare Struktur fest, mit welcher wir das Softwareprojekt durchführen würden. Er umfasst, welche Ziele wir mit dem Projekts erreichen wollen, wie diese erreicht werden sollen und gibt ein Zeitmanagement vor, um sie verwirklichen zu können. Wir haben uns bewusst gegen ein Vorgehensmodell wie Scrum entschieden, da der Projektumfang dafür zu klein ist und die Planung zu viel Zeit in Anspruch nehmen würde. Stattdessen entschieden wir uns, die Software iterativ zu entwickeln, wodurch flexibel auf die Wünsche des Kunden eingegangen werden kann. Des weiteren ermöglicht dieses Modell eine immer lauffähige Version des Produkts, was das Fehlerrisiko minimiert. Bei jeder Iteration wird an weiteren Funktionen und Verbesserungen des Produkts gearbeitet, um dem Kunden eine Software der besten Qualität ausliefern zu können. Nach jeder Iteration sendeten wir ein Softwarebuild an den Kunden, um ihn optimal in den Entwicklungsprozess einzubinden. Das ermöglichte uns rechtzeitig auf Änderungswünsche des Kunden reagieren zu können. Um sicher zu stellen, dass die Programmmodulen im Zeitplan realisiert, fertiggestellt und ausführlich getestet werden, haben wir uns für die Verwendung eines Kanban-Boards¹ entschieden.

Herr Steudle von der Firma Daimler TSS GmbH, bei welchem wir uns Feedback zu unserem Projektplan einholten, unterstützte und bekräftigte unsere Ideen mit positivem Feedback.

5.3 Problemanalyse

Schon bei der Vergabe der Projekte wurden Kernthemen der entsprechenden Softwareanforderungen von den Kunden angesprochen. Aus diesem Grund konnte bereits sehr früh an der Problemanalyse gearbeitet werden. Ein wichtiger Aspekt dabei war ein Shared Understanding mit dem Kunden zu schaffen, weshalb wir uns noch vor einem Treffen mit der Thematik Flow Design und mit Ralf Westphal ausgiebig auseinandergesetzt haben.

¹https://en.wikipedia.org/wiki/Kanban_board

5 Umsetzung

Aufgrund der ausgezeichneten Vorbereitung verlief das erste Treffen mit dem Kunden problemlos und es konnten entsprechende Anforderungen aufgestellt werden.

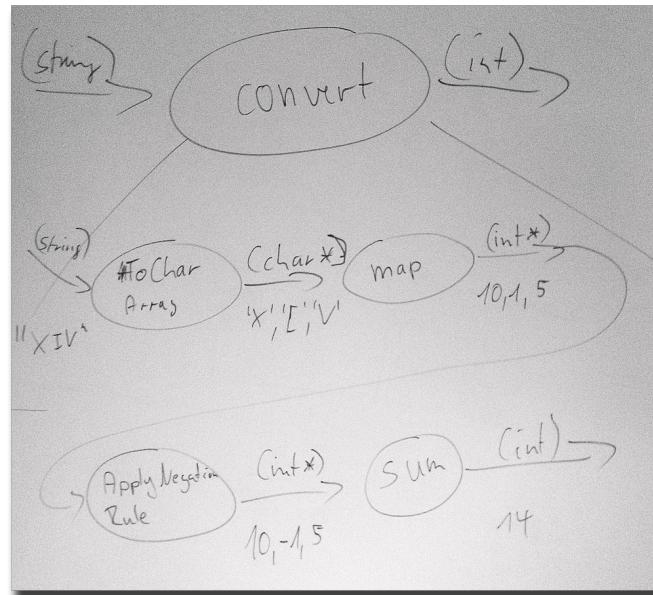


Abbildung 35: Problemanalyse Flow Diagramm

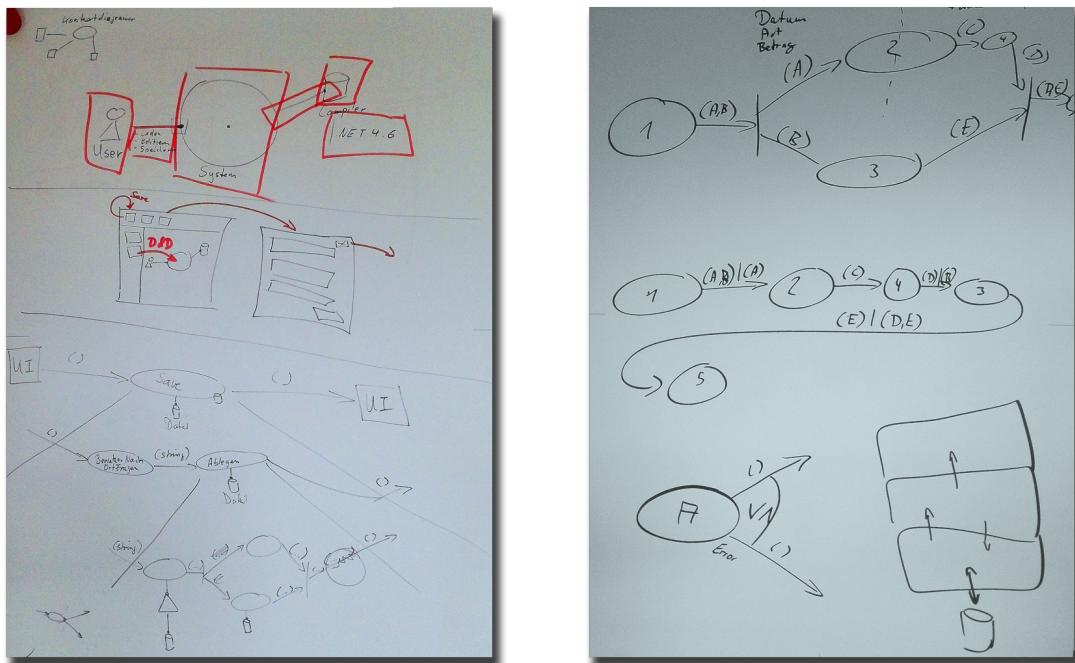


Abbildung 36: Problemanalyse beim Kunden am Flipchart

5.4 Spezifikation

Anhand der erarbeiteten Anforderungen erstellten wir eine Spezifikation (siehe Seite 6), welche mit dem Kunden durchgesprochen und überarbeitet wurde. Ein sehr großer Schwerpunkt bei der Spezifikation war das User Interface Design, da es den Kunden direkt betrifft. Das Design wurde sehr einfach und schlicht gehalten, wodurch eine intuitive Benutzung ermöglicht wird. Bei dem Design haben wir uns, sofern es möglich war, an die Gestaltungsgesetze nach Max Wertheimer² gehalten. Die einzelnen Designs befinden sich im Kapitel 4 (siehe Seite 18).

5.5 Systementwurf

Bei dem Systementwurf haben wir uns für eine Layered Architecture³ in C# entschieden, um die einzelnen Softwaremodule so gut wie möglich von einander trennen zu können. Dadurch können sie mit wenig Aufwand ausgetauscht oder erweitert werden. Diese Architektur ist die Basis unseres Codes, da die Software als Open Source Projekt realisiert wurde und deshalb auch für andere Teams und Mitwirkende klar strukturiert, einfach verständlich und leicht erweiterbar sein soll.

²<https://de.wikipedia.org/wiki/Gestaltpsychologie#Gestaltgesetze>

³https://en.wikipedia.org/wiki/Multilayered_architecture

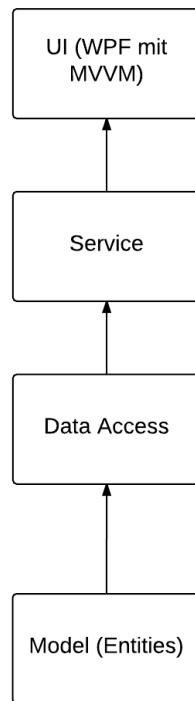


Abbildung 37: Layered Architecture

5.5.1 Model-Layer

Der Model-Layer enthält die Entities (Datenhaltende Klassen) unseres Systems. Er enthält keine Business Logik, die auf den Entities arbeitet. Dadurch erreichen wir eine Unabhängigkeit von Daten und der Logik, was dabei hilft die Logik auszutauschen ohne die Daten anfassen zu müssen.

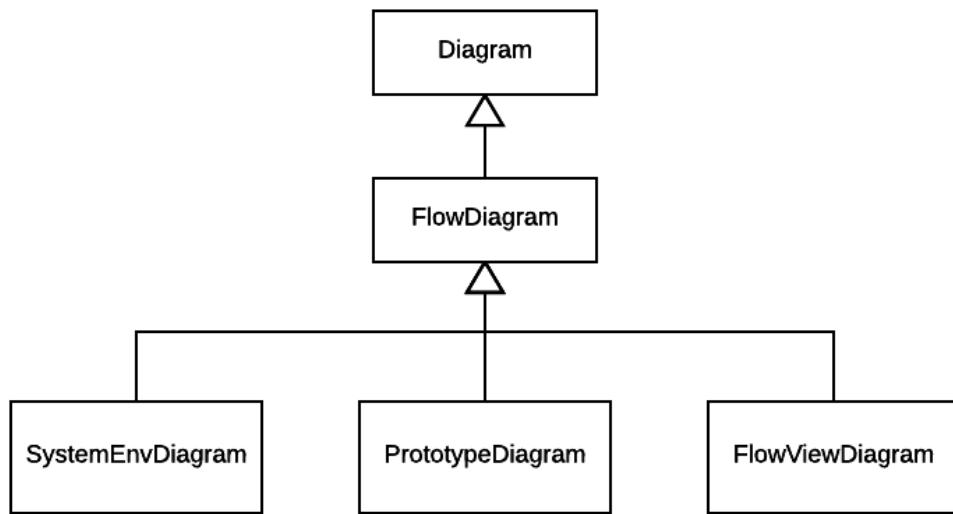


Abbildung 38: Hierarchie der Diagramme

Durch die Generalisierung mit *Diagram* ist es möglich die Anwendung mit weiteren Diagrammen zu erweitern.

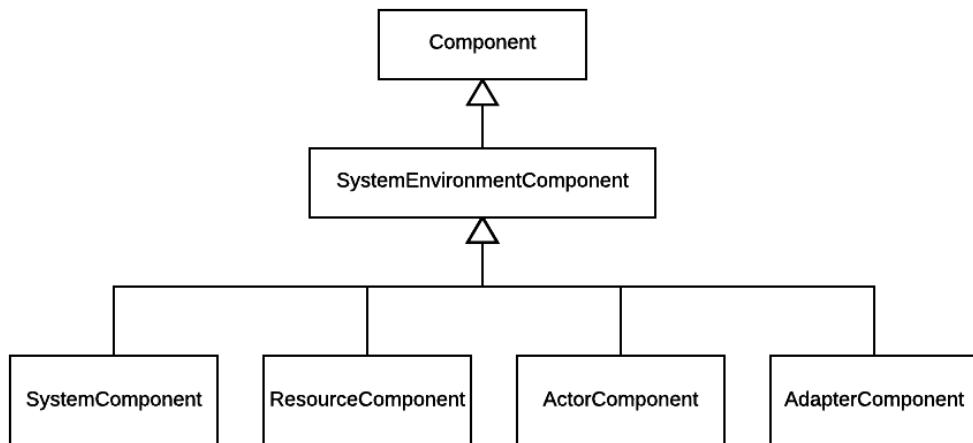


Abbildung 39: Komponenten des System-Umwelt-Diagramms

Abbildung 39 zeigt die Klassenstruktur die bei den Komponenten verwendet wurde. *Component* dient als Generalisierung um unterschiedliche Komponenten in einer gemeinsamen Liste speichern zu können ohne das beim hinzufügen von Komponenten diese geändert werden müssen.

den müssen. Die Klassenhierarchie für allgemeine, Maskenprototyp und FlowView Komponenten ist gleich aufgebaut. Die restlichen Entities existieren als einzelne Klassen (ohne Vererbung) im Model-Layer und müssen hier nicht extra behandelt werden.

5.5.2 Data Access-Layer

Der Data Access-Layer befindet sich an der Systemgrenze und bildet damit die Kommunikation zu externen Datenquellen, wie zum Beispiel Datenbanken oder das Filesystem. Das ermöglicht uns eine austauschbare und einfach erweiterbare Kommunikation nach „außen“, wobei egal ist, ob die Datenquelle ein Filesystem, eine Datenbank oder ein Webservice ist. Außerdem enthält der Layer einen Converter, mit welchem die Objektdaten in eine gewünschte Form, wie zum Beispiel XML, konvertiert werden können.

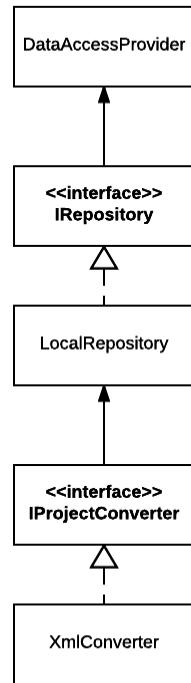


Abbildung 40: Aufbau des Data Access-Layers

Während das *LocalRepository* als Schnittstelle zu externen Datenquellen dient, sorgt der *DataAccessProvider* dafür, dass ein geeignetes Repository(Filesystem, Datenbank...) für die anderen Layer zur Verfügung steht.

5.5.3 Service-Layer

Der Service-Layer enthält die eigentliche Logik und arbeitet mit den Klassen aus dem Data Access-Layer und dem Model-Layer. Er ist zudem für die Erstellung von Diagrammen und Diagramm Komponenten zuständig, da hierfür auch eine Validierung (z.B. keine Diagramme mit gleichen Namen) notwendig ist. Die konkreten Implementierungen der Service Klassen sind außerhalb des Service-Layers nicht sichtbar (C# Sichtbarkeit „internal“). Eine Klasse, die die Implementierungen kennt, erstellt bei Bedarf eine konkrete Implementierung und gibt sie als Interface nach außen. Somit stellen wir sicher, dass wir Implementierungen austauschen können ohne die Lauffähigkeit des Systems zu gefährden.

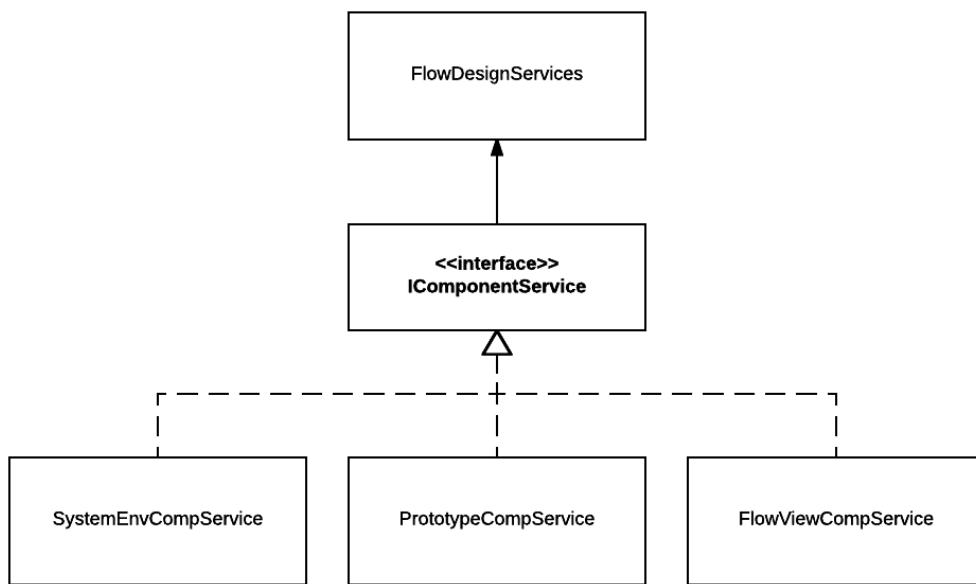


Abbildung 41: Teilklassendiagramm des Service Layers

In Abbildung 41 ist ein Ausschnitt aus dem Service-Layer zusehen um das Prinzip zu verdeutlichen. Alle Services die eine Komponente erstellen, implementieren das Interface *IComponentService*, das außerhalb des Service-Layers sichtbar ist. Die konkreten Implementierungen *SystemEnvCompService*, *PrototypeCompService* und *FlowViewCompService* werden bei Bedarf von *FlowDesignServices* instanziert und sind außerhalb nicht sichtbar. *FlowDesignServices* gibt die Instanz des Services nun als Interface zurück und dient so als Schnittstelle zu einem anderen Layer. Alle anderen Services funktionieren nach dem selben Prinzip und nutzen ebenfalls die *FlowDesignServices* Klasse als Schnittstelle zu anderen Layern.

5.5.4 UI-Layer

Der UI-Layer entählt Controls, Resourcen (Bilder, Farben, Icons...) die notwendig sind um die Oberfläche und Diagramme darzustellen. Dies wurde mit dem Entwurfsmuster „Model View ViewModel“⁴ erreicht. Die UI ist so aufgebaut, dass es nur ein Fenster gibt, das die Grundlegenden Elemente wie Toolbars, Statusleiste und ein Frame enthält. Alle anderen Ansichten sind als Pages realisiert und werden im eben genannten Frame dargestellt. Durch die Aufteilung der einzelnen Ansichten ist es möglich die UI nach belieben zusammen zubauen. So wurde die Startseite aus drei Pages erstellt, der eigentlichen Startseite, einer Infoseite und einer Page um ein Projekt zu erstellen. Die Startseite enthält wiederum ein Frame um die Infoseite und die Page für die Projekterstellung anzuzeigen.

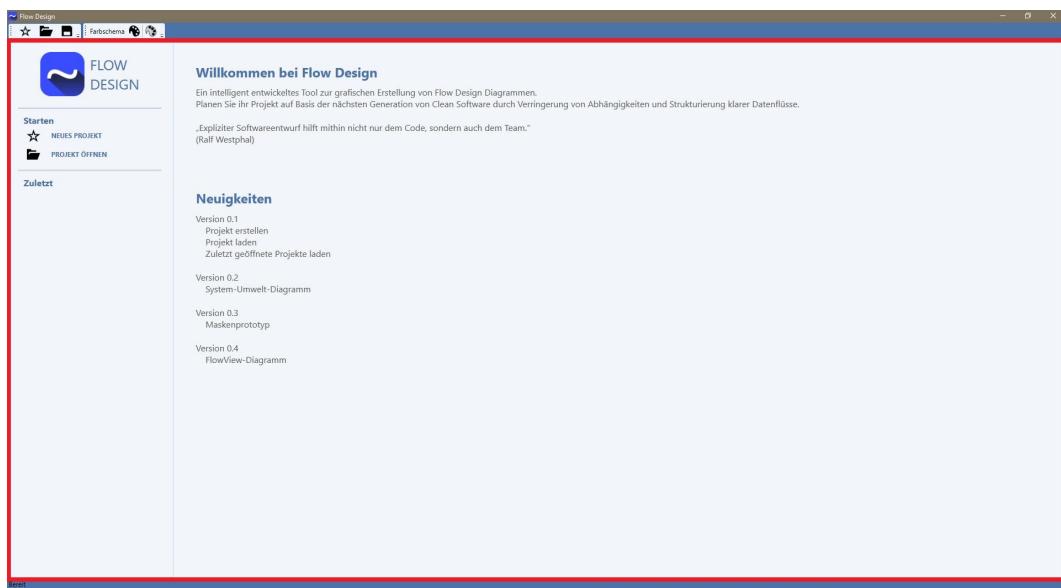


Abbildung 42: Fenster mit Startseite

Der rot makierte Bereich ist der Frame unseres einzigen Fensters, der die Page für die gesamte Startseite anzeigt.

⁴https://de.wikipedia.org/wiki/Model_View_Model

5 Umsetzung

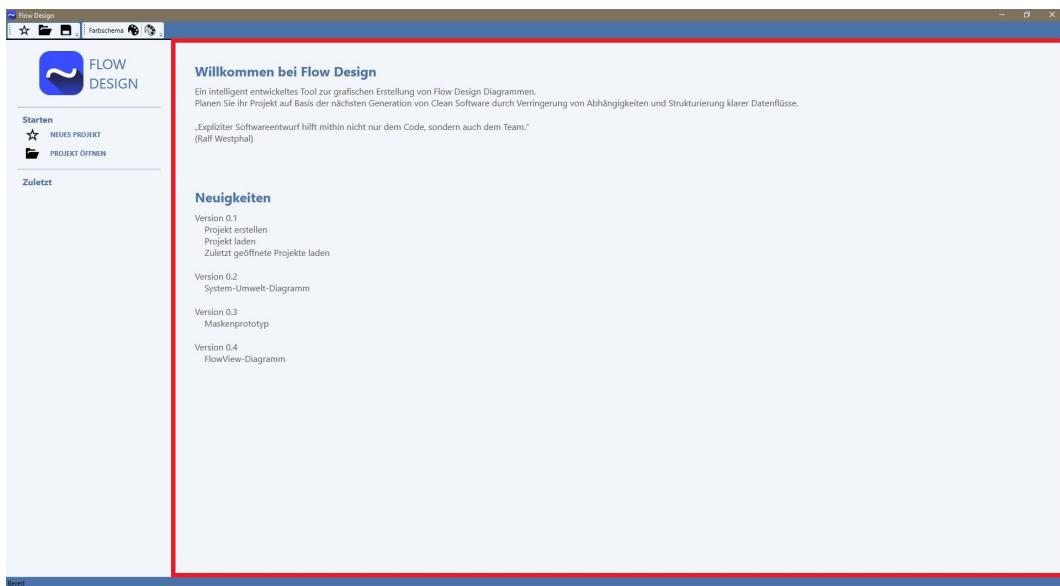


Abbildung 43: Startseite mit Infoseite

Die Startseite enthält wie oben beschrieben ebenfalls ein Frame um Pages anzuzeigen (in Abbildung 43 rot markiert). In diesem Fall hält der Frame die Page für die Infoseite.

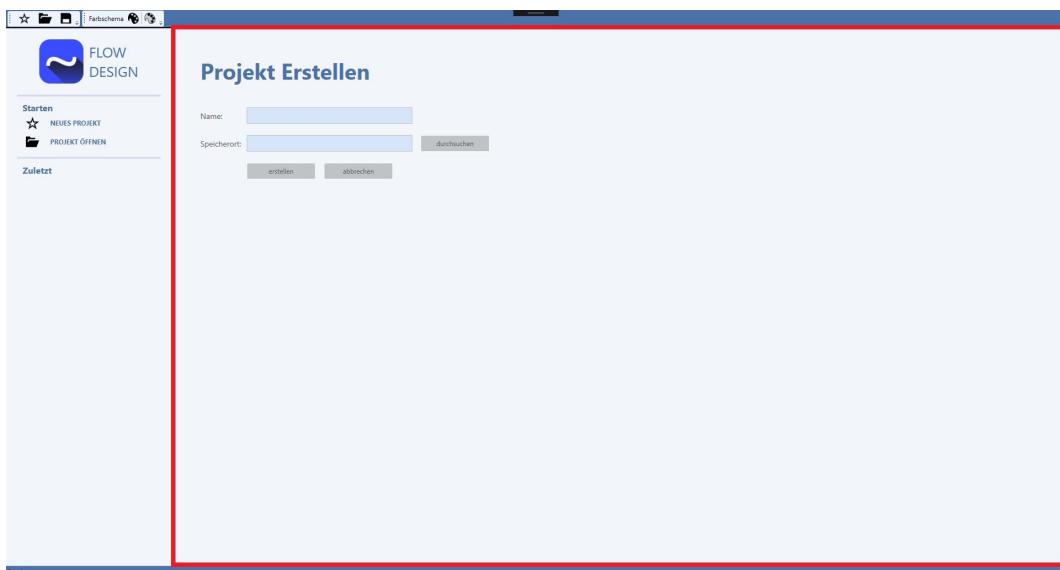


Abbildung 44: Startseite mit Projekterstellung

In Abbildung 44 wurde die Infoseite durch die Projekterstellung ersetzt. Ohne die Aufteilung in die einzelnen Pages, wäre es nötig gewesen immer wieder Fenster zu öffnen, was die Anwendung weniger übersichtlich macht und den Benutzer bei zu vielen Fenstern

überfordern kann. Zudem hilft es bei der Entwicklung, da an unterschiedlichen Ansichten gearbeitet werden kann, die auf dem selben Fenster dargestellt werden, ohne dass sich die Entwickler gegenseitig stören.

5.6 Implementierung

Wie im Projektplan erwähnt, entwickelten wir die Software mit dem iterativen Vorgehensmodell. Im Gegensatz zum spezifikationsorientierten Modell, bei dem das Testen erst am Ende des Projekts stattfindet, wird bei dem interativen Modell nach jeder Modulentwicklung oder Änderung getestet. Durch das frühzeitige Testen können Folgefehler, die aufgrund von unentdeckten Fehlern entstehen, vermieden werden, was die Qualität der Software erheblich steigert. Ein weiterer Vorteil ist die Flexibilität um Anpassungen vorzunehmen, da sich die Anforderungen des Kunden bereits während der Entwicklung ändern/erweitern können. Um während der Implementierung den Überblick über die zu entwickelnden Module gewährleisten zu können, erstellten wir ein Kanban-Board. Dort gibt es für jeden Entwicklungsschritt eine Spalte, wobei die Anzahl der zulässigen Karten pro Spalte begrenzt wird. Dadurch erreichten wir, dass Module zeitnah fertiggestellt und getestet werden, bevor mit neuen Modulen begonnen werden kann.

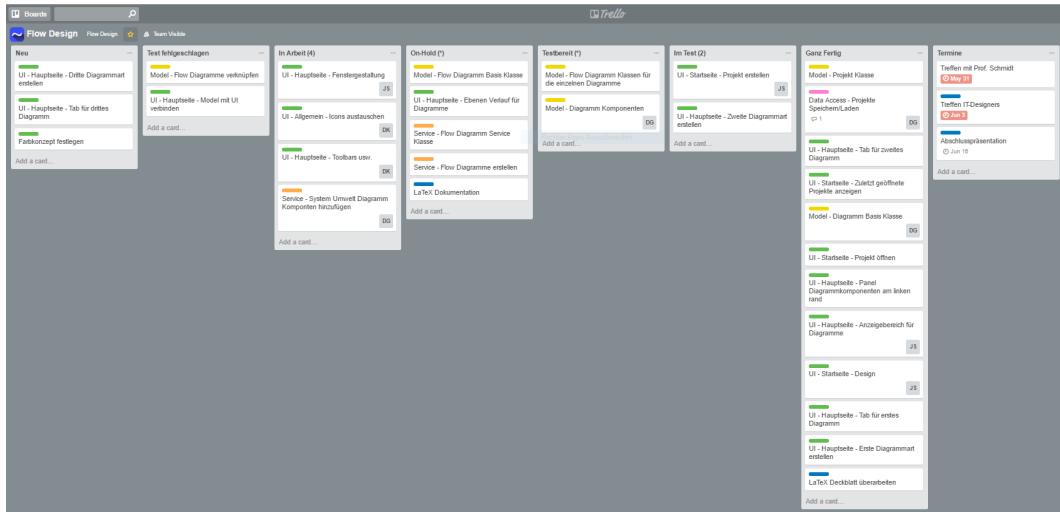


Abbildung 45: Kanban Board Trello



Abbildung 46: Kanban Board Detail

Bei der Programmierung teilten wir jedem Team-Mitglied eine bestimmte Verantwortlichkeit zu:

- Johannes Steinhülb - Entwicklungsleiter und Frontend
- Daniel Kratzel - Application Design
- Daniel Glinka - Backend
- Johannes Schlier - Testing

Aufgrund eines hervorragenden Shared Understanding und der guten Zusammenarbeit zwischen den einzelnen Team-Mitgliedern war eine bereichsübergreifende Entwicklung möglich, was den Entwicklungsprozess erheblich beschleunigte. Bei regelmäßigen Treffen mit dem Kunden präsentierten wir unseren Fortschritt und konnten Unklarheiten, welche während der Implementierung aufgetreten sind, lösen.

5.6.1 Datenspeicherung

Alle Diagramme/die Projekt-Info sind in einer eigenen XML-Datei serialisiert, wobei der Inhalt der Datei die Datenstruktur in der Anwendung widerspiegelt. Zum Serialisieren/De-

Serialisieren wurde der XML-Serializer von C# verwendet. Die XML-Dateien werden in einem Ordner abgelegt, der den Namen des Projekts erhält. Die Namen der Dateien entsprechen den angegebenen Diagrammnamen und Diagrammtypen.

5.7 Test

Um jederzeit eine lauffähige Version sicherzustellen und um die Qualität der Software zu optimieren wurden nach jeder Änderung in dem Programm dessen Funktionalität getestet. Geschehen ist dies sowohl in Manuellen Tests, in denen die jeweils aktuelle Version gestartet wurde und alle möglichen Aktionen ausgeführt wurden, als auch in automatisierten Unit Tests. Für die Backend Services wurden die Unit Tests geschrieben, in denen die Funktionalität überprüft und über Asserts der Test als bestanden oder durchgefallen markiert wird. Dabei wurden auch Randfälle abgefragt. Verwendet wurde hierzu die Microsoft Unit Testing Bibliothek.

Da die UI Klassen nur die Backend Services aufrufen und selber keine große Logik enthalten, wurden diese manuell überprüft und mögliche Randfälle so überprüft.

Es wurden damit einige Bugs gefunden, die gemeldet, und danach gefixt wurden.

5.8 Versionsverwaltung

Für die Versionsverwaltung haben wir Git⁵ gewählt, da es eine Integration in der von uns genutzten IDE Visual Studio gibt. Die Hauptarbeit wurde auf dem dev-Branch durchgeführt, um spätere Überschneidungen mit den Release-Banches zu verhindern. Wurden alle Features für einen Release implementiert, haben wir den abschließenden Commit des Releases mit einem Tag versehen. Danach wurde für den Release ein extra Branch erstellt um die Übersichtlichkeit im Repository zu erhöhen und einfacher zu einem Release springen zu können.

Für die Commit Nachrichten selbst haben wir einen Styleguide eingeführt, um die Änderungen besser nachverfolgen zu können. Der Styleguide sieht wie folgt aus:

Änderungsart - Was wurde geändert/hinzugefügt/gelöscht, wobei die Änderungsart „Add“, „Change“ oder „Remove“ sein kann.

⁵<https://de.wikipedia.org/wiki/Git>

5.9 Zeitmanagement

Der Zeitplan des Projekts ist mit den Meilensteinen (Seite 16) beschrieben. Diesen konnten wir während des kompletten Projekts einhalten und konnten es somit rechtzeitig fertigstellen. Rückblickend mussten wir jedoch feststellen, dass wir trotz der Verwendung eines Kanban-Boards etwas unter Zeitdruck kamen, da wir am Anfang viel Zeit dafür benötigt haben, ein allgemeines Diagrammframework zu erstellen. Um diesem Problem entgegen zu wirken werden wir in Zukunft für die einzelnen Implementierungsschritte feinere Meilensteine und Zeitvorgaben festlegen.

6 Produkt

Das Produkt Flow Design ist ein Prototyp, welcher ein Tool zum Erstellen von Flow Design Diagrammen darstellt. Diese Diagramme sind in dem Buch „The Architect’s Napkin“ [2] von Ralf Westphal beschrieben.

6.1 User-Guide

6.1.1 Einleitung

Das folgende Kapitel beschreibt die Grundlegenden Use Cases des Programms und wie diese konkret ausgeführt werden.

6.1.2 Erstellen eines neuen Projekts

Öffnet man das Programm, sieht man als erstes die Startseite. Auf der linken Seite der Startseite gibt es einen Menüeintrag zum Erstellen eines neuen Projekts.
Klickt man hierauf, müssen ein Name für das Projekt, sowie ein Speicherort dafür ausgewählt werden. Nur, wenn diese Korrekt vergeben werden, wird das Projekt erstellt.



Abbildung 47: Schritt 1: „Neues Projekt“ auswählen.

6 Produkt

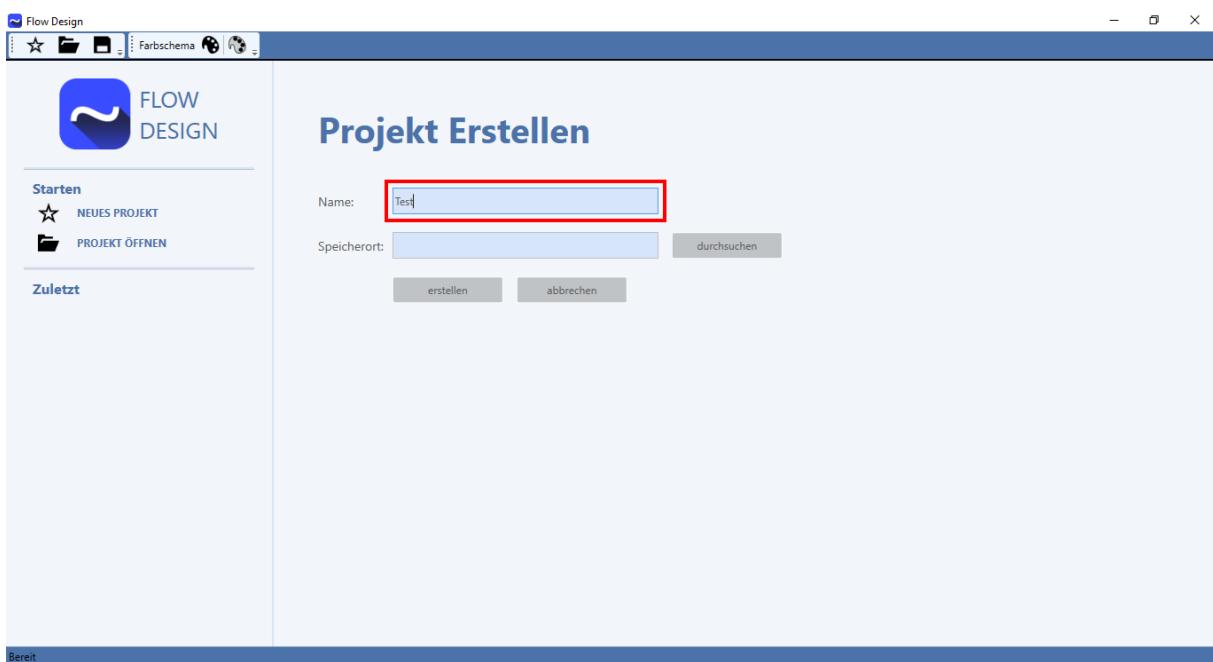


Abbildung 48: Schritt 2: Dem Projekt ein Namen geben.

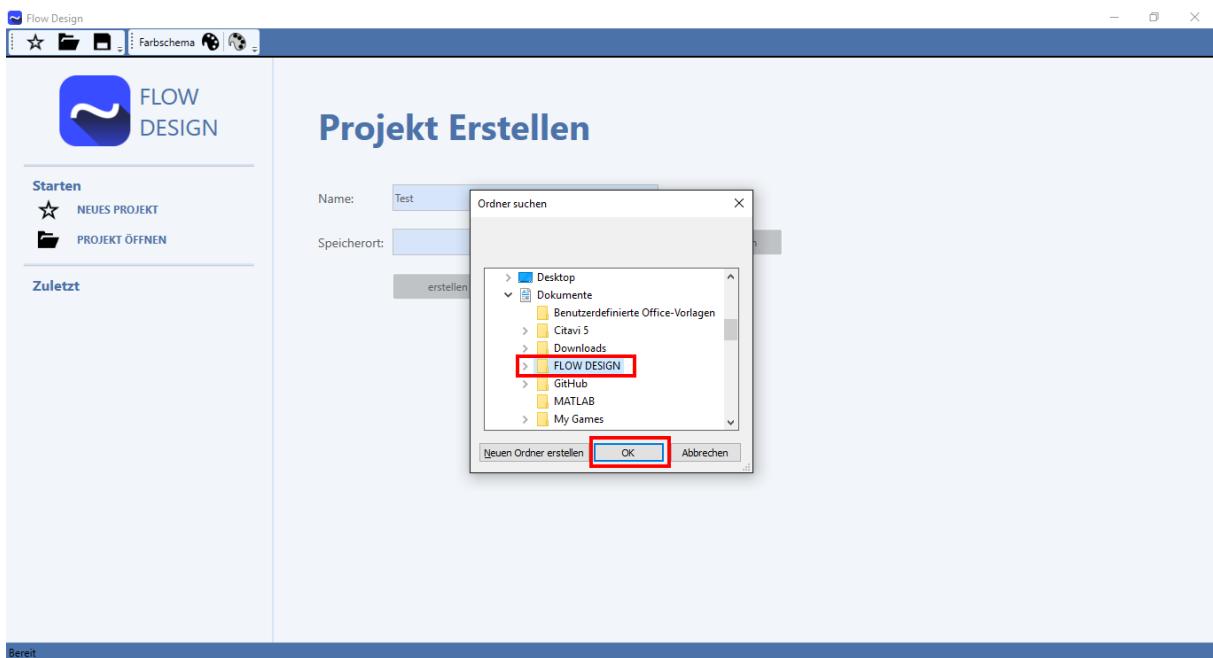


Abbildung 49: Schritt 3: Über „durchsuchen“ wird ein Ordner zum Speichern ausgewählt und mit „OK“ bestätigt.

6 Produkt

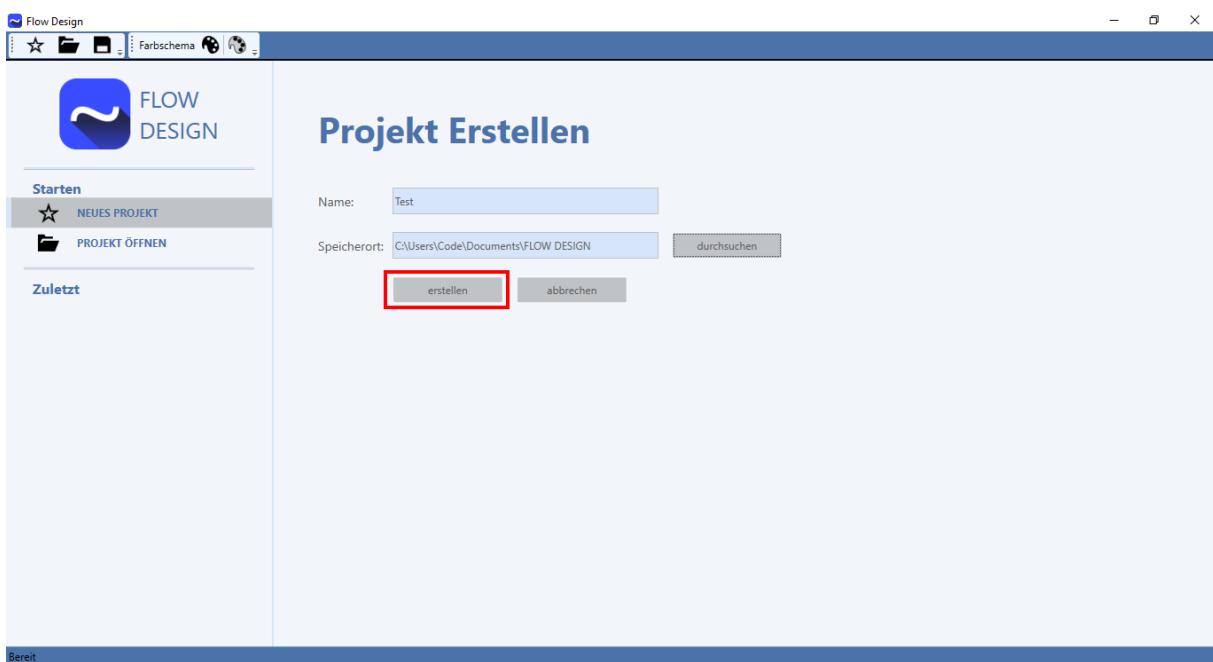


Abbildung 50: Schritt 4: Über „erstellen“ wird das Projekt angelegt.

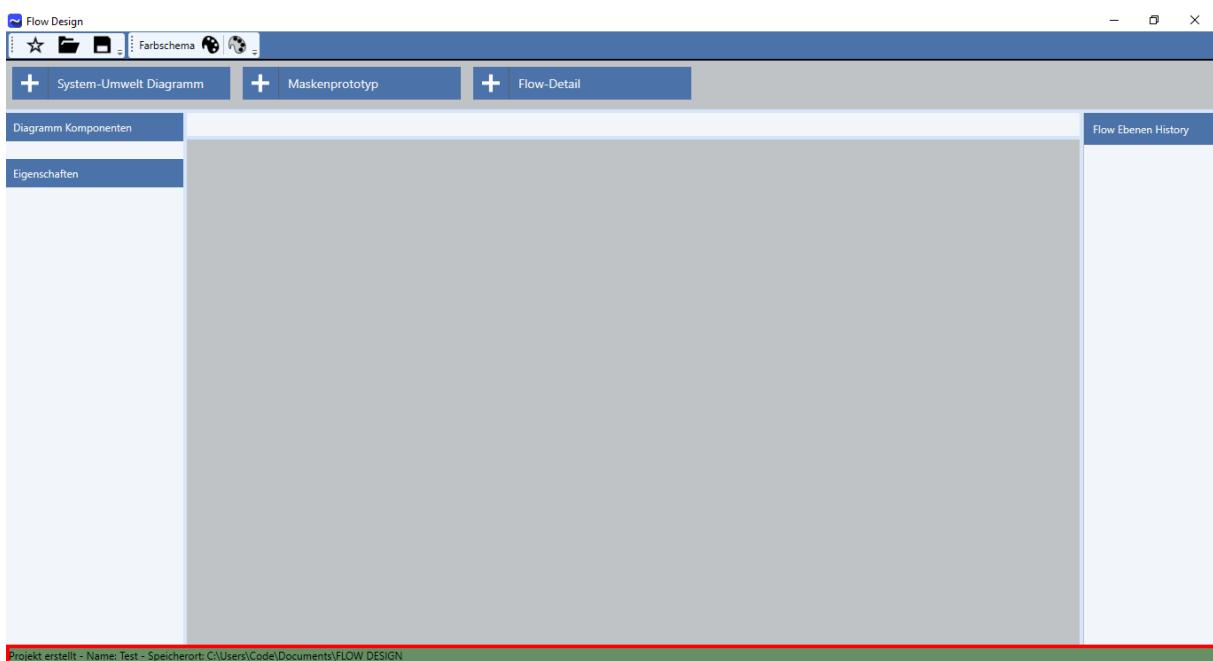


Abbildung 51: Schritt 5: In der Statusleiste wird nach erfolgreichem Erstellen des Projekts dessen Name und Speicherort angegeben.

6.1.3 Speichern eines Projektes

Nach jeder Änderung eines Diagramms wird das Projekt automatisch gespeichert. Zudem kann es noch explizit über den Speichern-Menüeintrag gespeichert werden. Der Speicherort der Diagramme ist der beim Erstellen eines Projekts angegebene Projektordner.

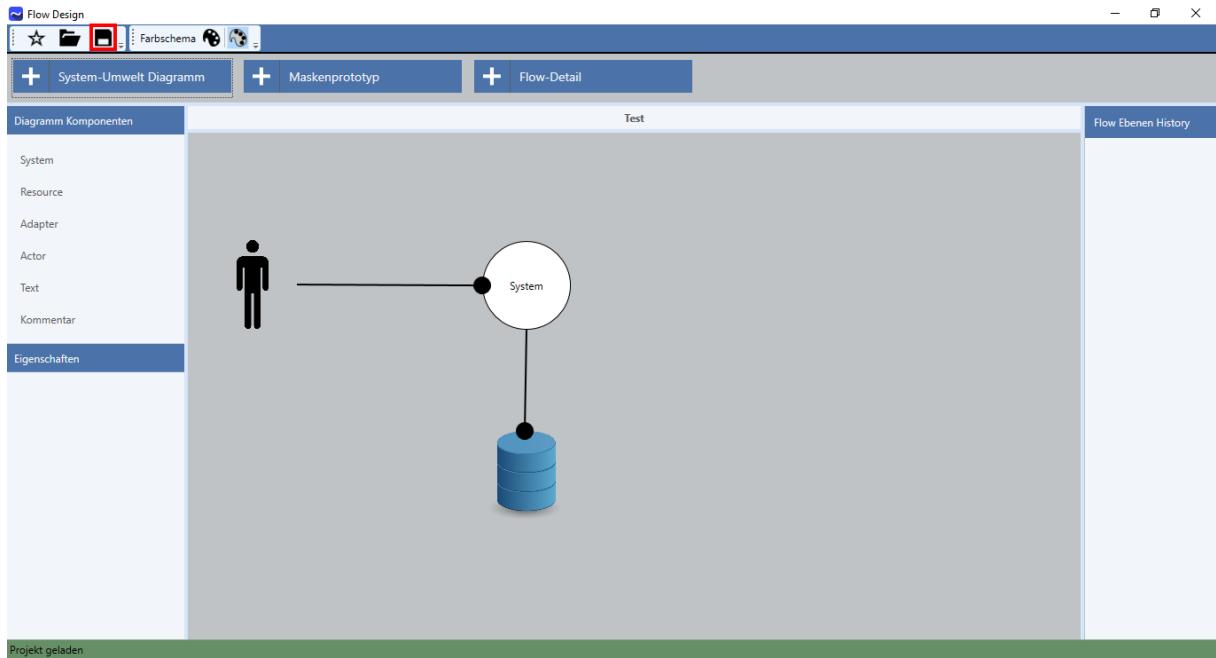


Abbildung 52: Schritt 1: Um die Änderungen an dem Aktuellen Projekt manuell zu speichern, klickt man in der Menüleiste auf das „Speichern“ Symbol.

6.1.4 Öffnen eines vorhandenen Projekts

Ein vorhandenes Projekt kann entweder über die Auswahl bei den zuletzt verwendeten Projekten oder über den Menüeintrag „öffnen“ geöffnet werden. Wenn der Menüeintrag gewählt wird, öffnet sich ein Dialog, in dem das gewünschte Projekt ausgewählt werden kann. Dazu muss im Dialog die „flow-Datei“ ausgewählt werden. Sollte beim Laden etwas nicht funktionieren wird eine entsprechende Fehlermeldung ausgegeben und der Vorgang wird abgebrochen. Ist das nicht der Fall wird das gewünschte Projekt geladen.

6 Produkt



Abbildung 53: Schritt 1: Aus der Startseite heraus können über mehrere Schaltflächen Projekte geöffnet werden. Klickt man auf eines der Projekte in der Liste der zuletzt verwendeten Projekte, wird es direkt geöffnet. Über die „Projekt öffnen“ Schaltflächen Links und im Menü am oberen Rand, muss ein Projekt von der Festplatte ausgewählt werden.

6 Produkt

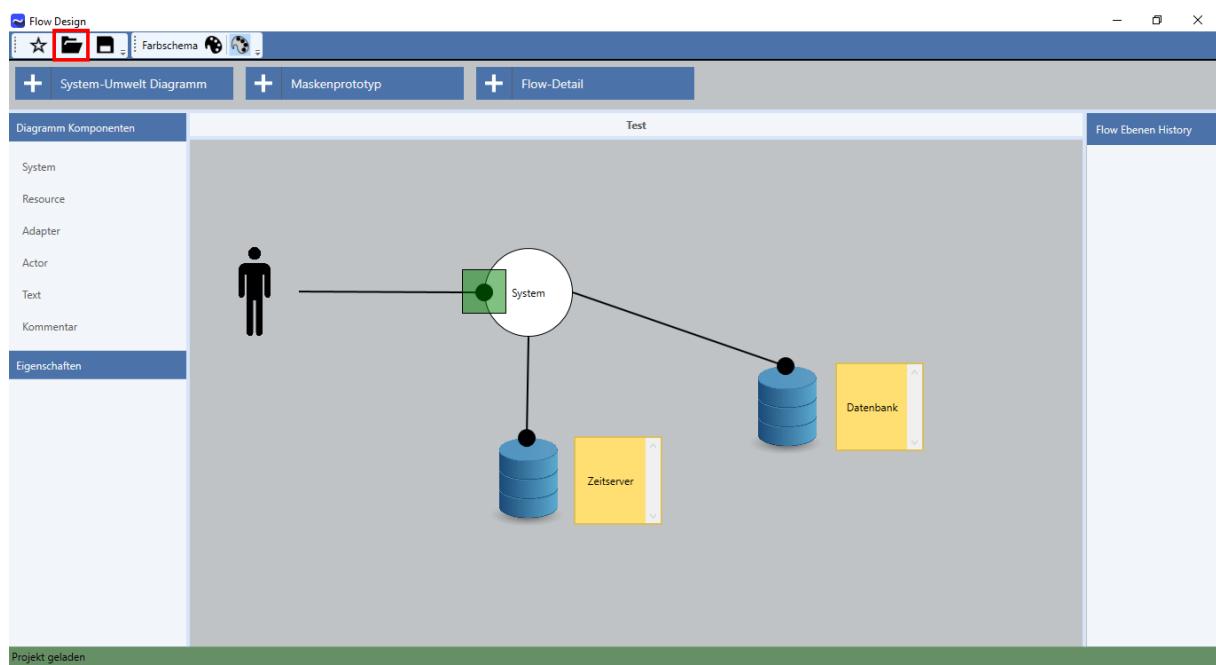


Abbildung 54: Schritt 5: Aus der Diagrammansicht kann die „Projekt öffnen“ Schaltfläche verwendet werden.

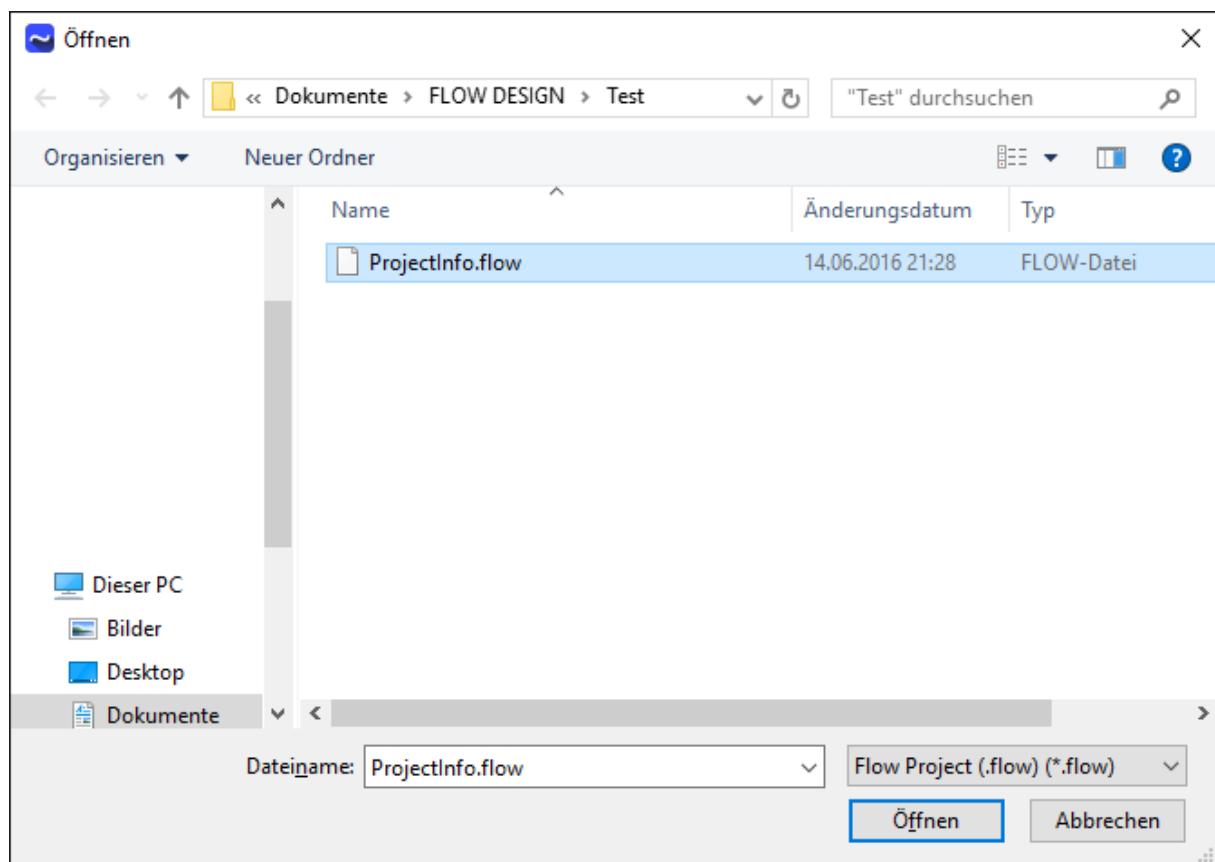


Abbildung 55: Schritt 3: Um ein Projekt von der Festplatte zu laden, muss man in dessen Speicherpfad gehen, die „.flow“ Datei auswählen und öffnen.

6.1.5 Erstellen eines Diagramms

Um ein Diagramm zu erstellen, muss zunächst der gewünschte Diagrammtyp innerhalb eines Projekts ausgewählt werden. In dem dafür vorgesehenen Feld muss der Name des zu erstellenden Diagramms angegeben werden. Das gewünschte Diagramm erscheint nun in der Dropdownliste, von der es zur Ansicht/Bearbeitung ausgewählt werden kann.

6 Produkt

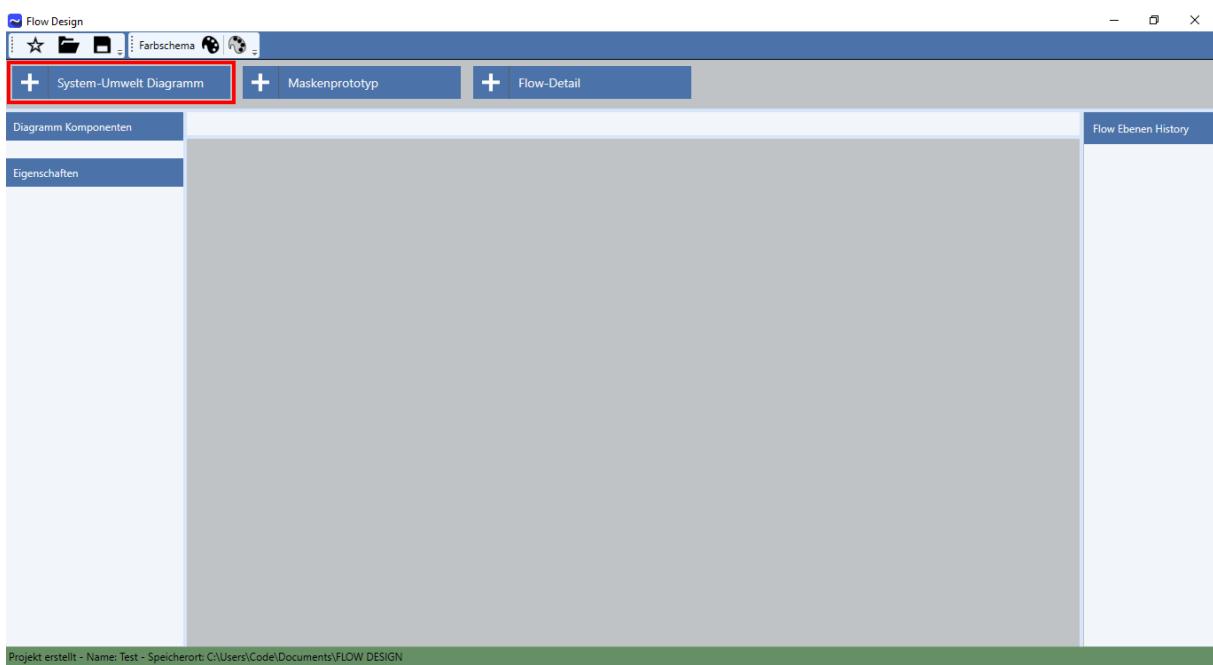


Abbildung 56: Schritt 1: Über einen der drei Diagramm Buttons das Dropdown-Menü öffnen.

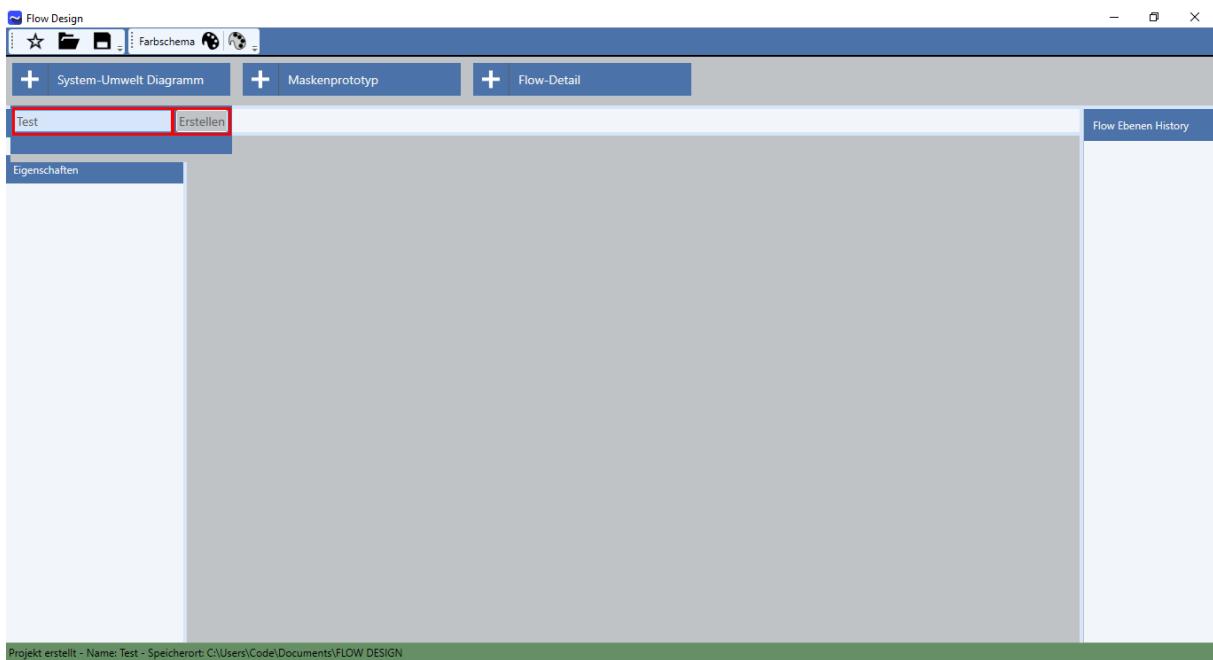


Abbildung 57: Schritt 2: Einen Namen eingeben und auf „Erstellen“ klicken.

6 Produkt

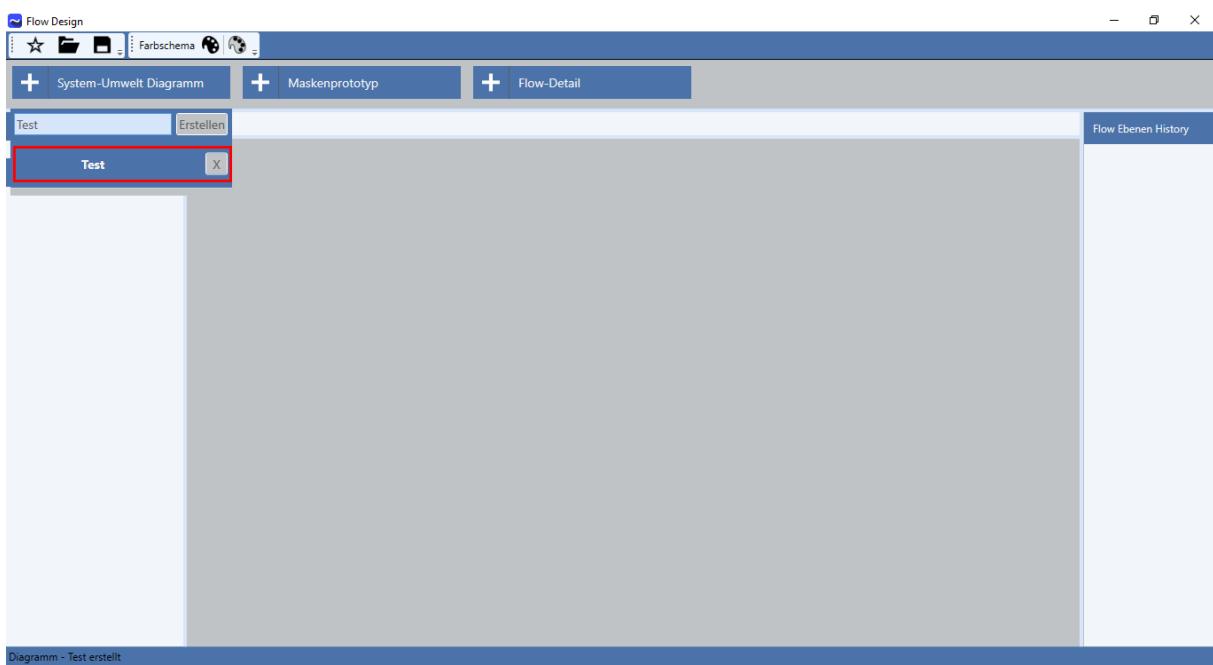


Abbildung 58: Schritt 3: Das Diagramm wurde erstellt und kann nun ausgewählt werden.

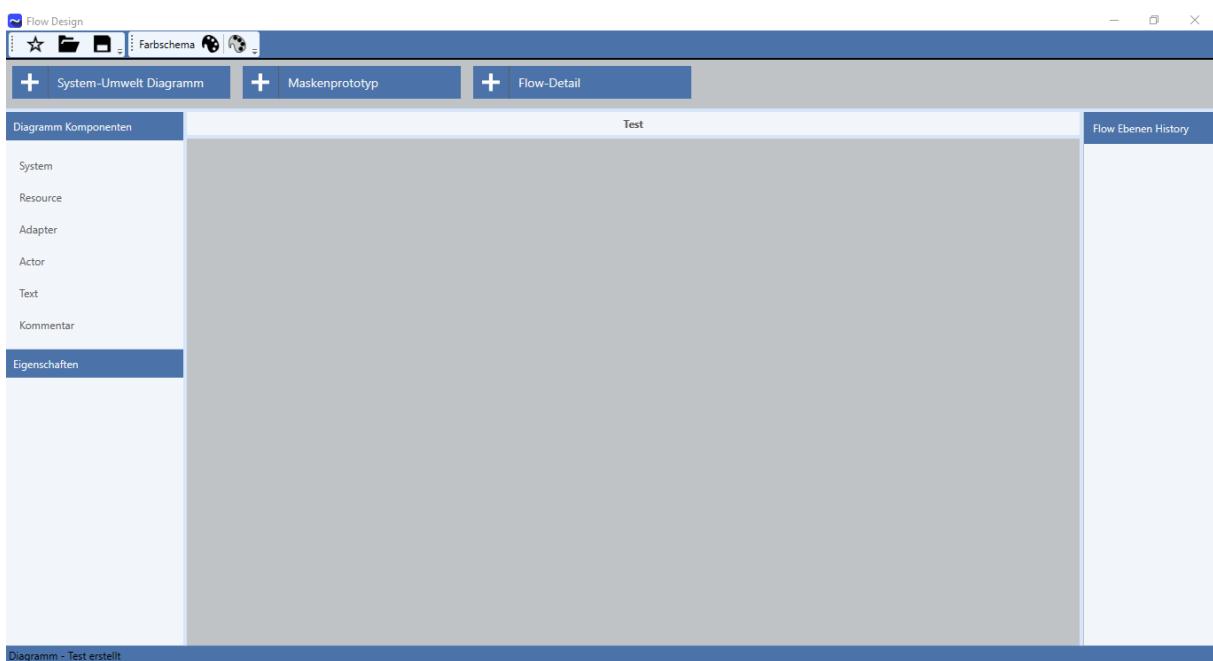


Abbildung 59: Schritt 4: Die Ansicht zum Bearbeiten des Diagramms.

6.1.6 Diagramm bearbeiten

Um ein bereits erstelltes Diagramm zu bearbeiten muss sich bereits in dem entsprechenden Projekt befinden (siehe 6.1.4). Nun wählt man im Menü den gewünschten Menüpunkt aus und kann in der Dropdownliste das zu bearbeitende Diagramm auswählen. Die einzelnen Komponenten können nun per Drag and Drop von der linken Sidebar in das Diagramm gezogen werden. Einzelne Einstellungen können im Einstellungsmenü unter der Sidebar angepasst werden. Mehrere Komponenten können miteinander verbunden werden. Dazu bewegt man den Cursor über die gewünschte Startkomponente, bis an allen vier Seiten sogenannte „Verbinder“ erscheinen. Nun klickt man den Verbinder der Startkomponente an und bewegt den Cursor auf einen Verbinder einer Zielkomponente und lässt die Maustaste wieder los. Die zwei Komponenten sind anschließend durch eine Verbindungsline verbunden. Mit der rechten Maustaste können Komponenten und Verbindungen wieder entfernt werden.

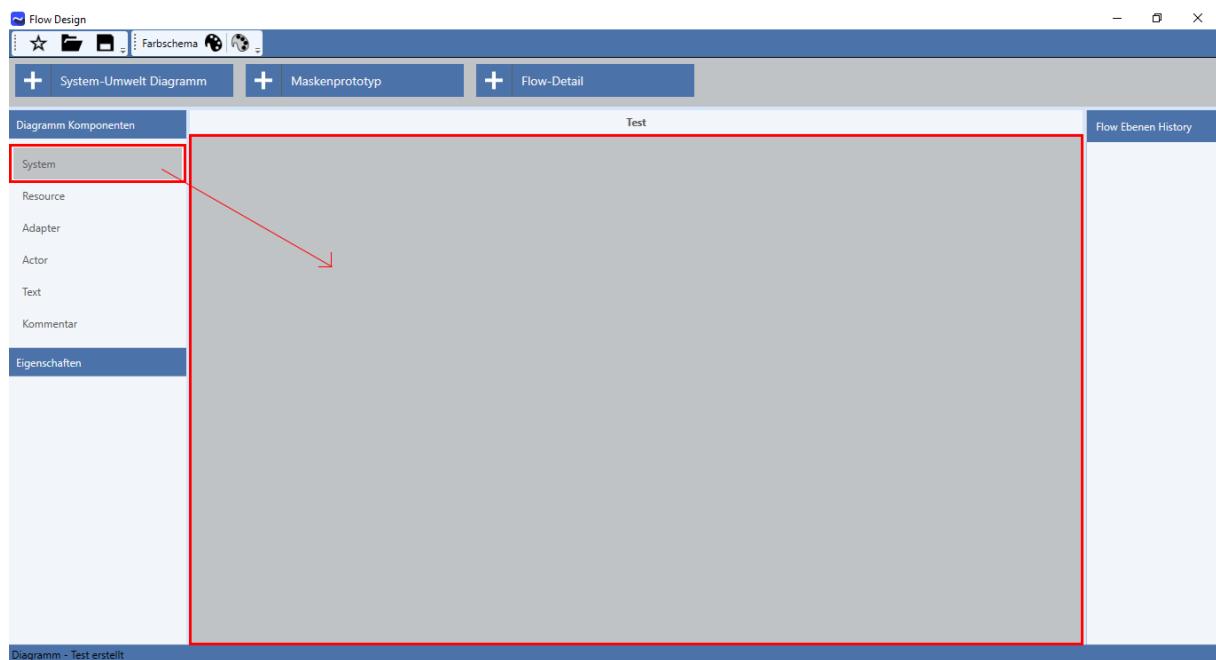


Abbildung 60: Schritt 1: Eine Komponente von links per Drag and Drop in das Diagramm Feld ziehen.

6 Produkt

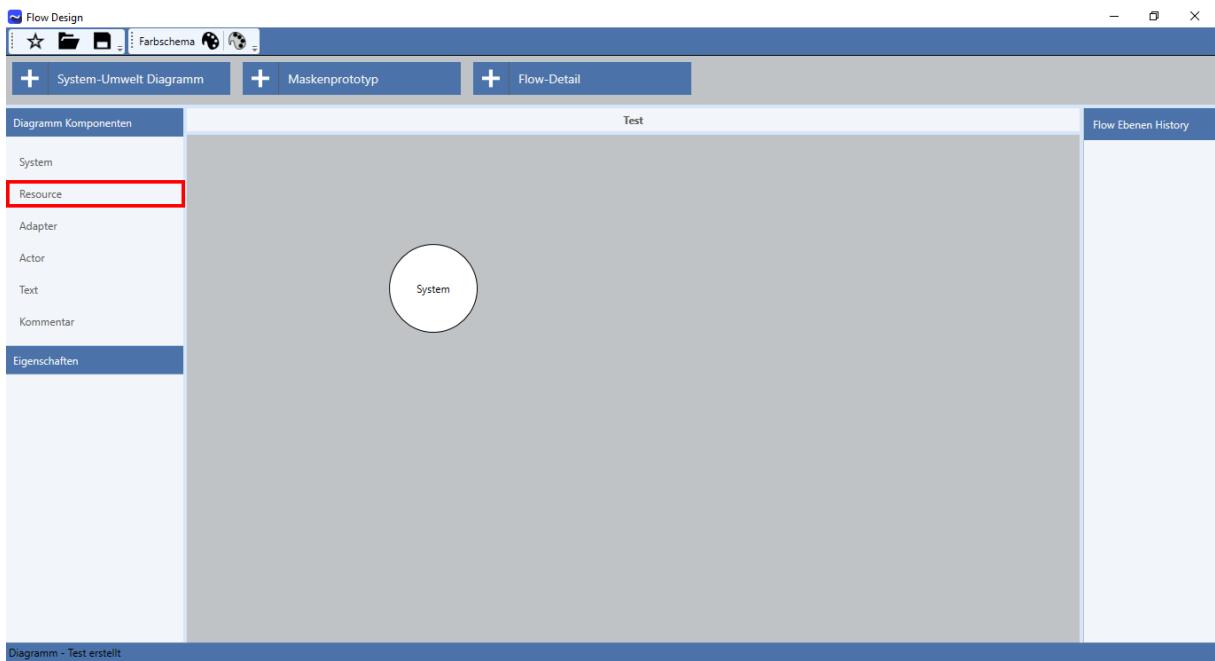


Abbildung 61: Schritt 2: Es können nun weitere Komponenten hinzugefügt werden (z.B. eine Ressource).

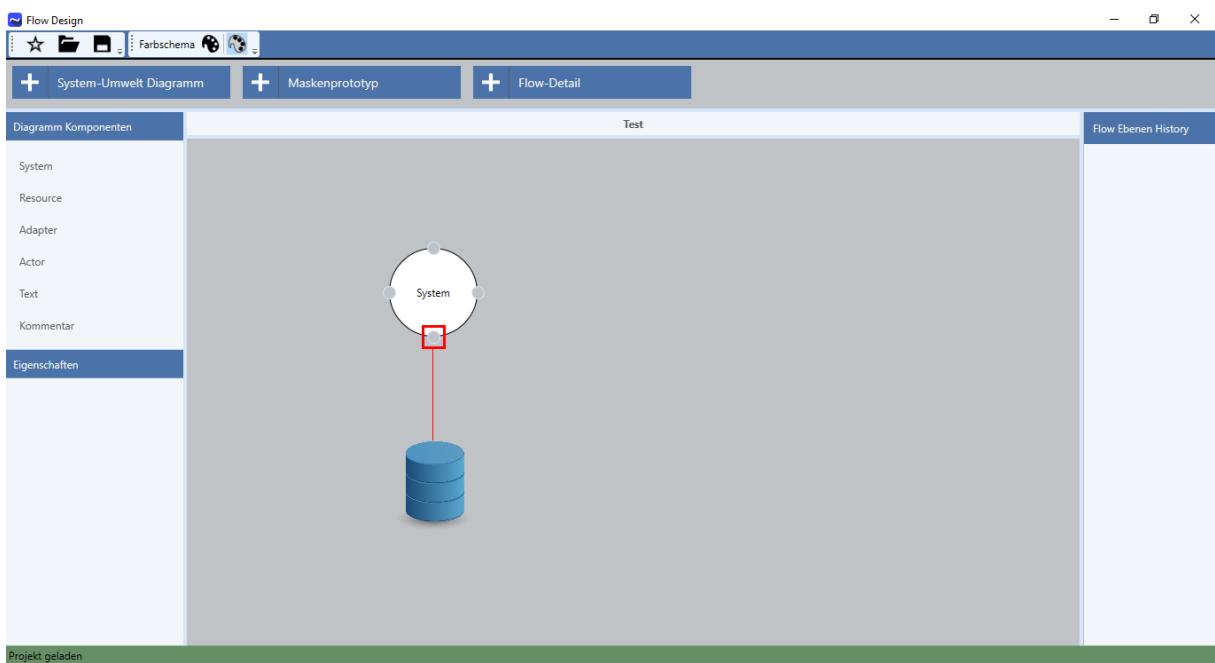


Abbildung 62: Schritt 3: Klickt man auf einen Konnektor, kann durch gedrückt halten der linken Maustaste und Ziehen auf einen weiteren Konnektor eine Abhängigkeit auf ein Anderes System hinzugefügt werden. Das System von dem man die Abhängigkeit aus zieht, ist von dem Ziel abhängig.

6 Produkt

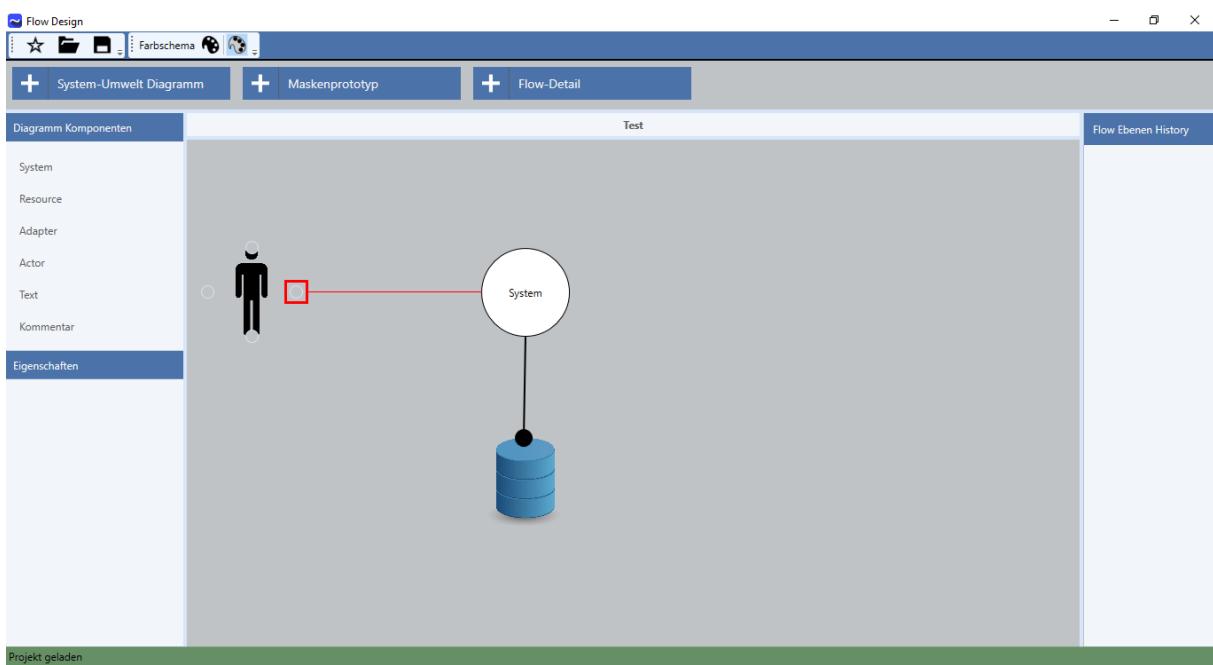


Abbildung 63: Schritt 4: Auf gleiche Weise können auch Akteure hinzugefügt werden. Diese sind im Normalfall vom System abhängig, also werden die Verbinden vom Akteur aus gezogen.

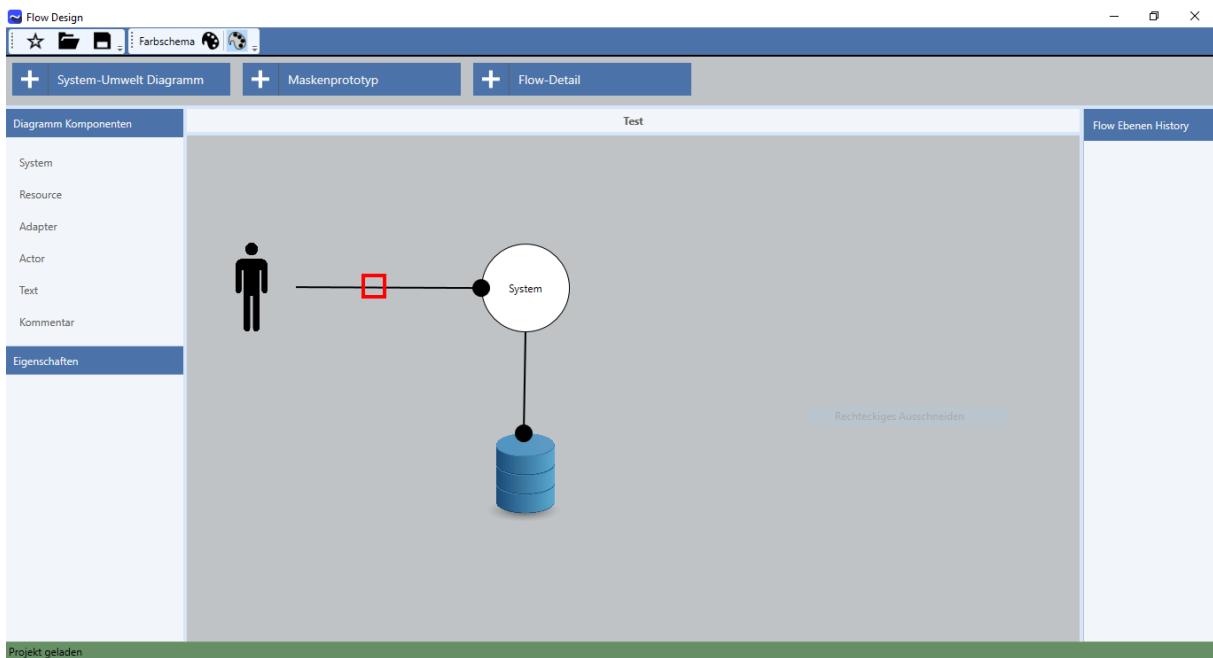


Abbildung 64: Schritt 5: Um eine Abhängigkeit zu Löschen, klickt man mit der Rechten Maustaste auf sie.

6.1.7 Diagramm löschen

Diagramme können gelöscht werden, indem im Menüband der entsprechende Diagrammtyp wie beim Bearbeiten eines Diagramms (siehe 6.1.6) ausgewählt wird. Nun kann in der Dropdownliste ein Diagramm über die Schaltfläche rechts neben dem Diagrammnamen gelöscht werden.

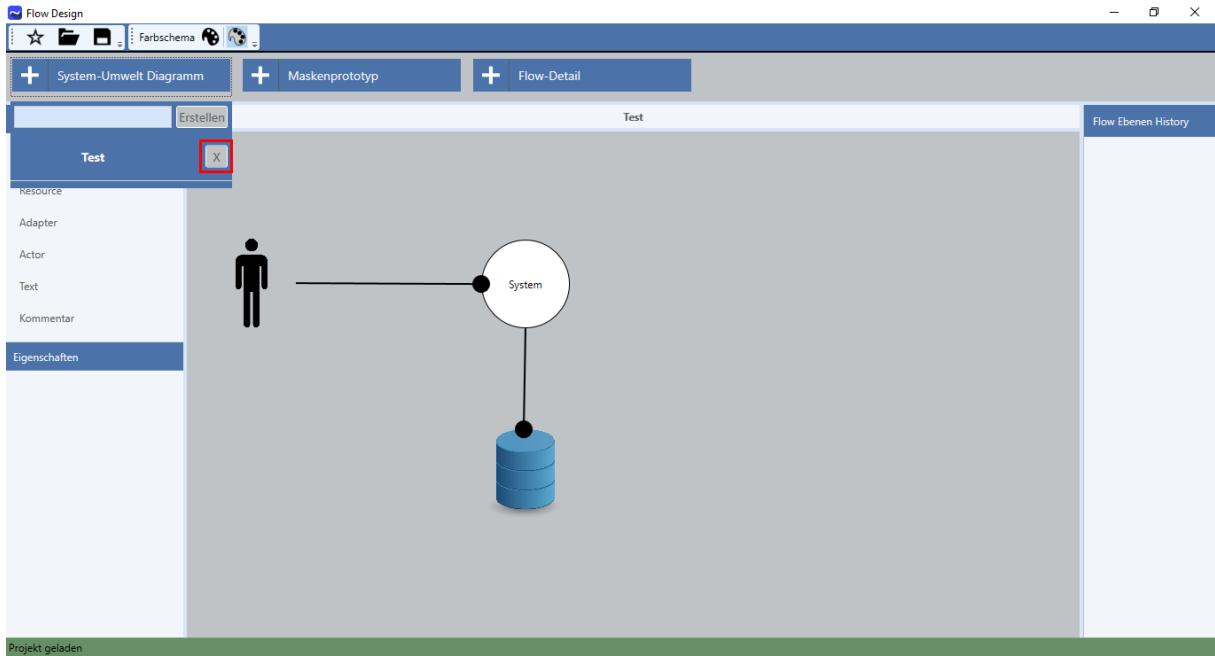


Abbildung 65: Schritt 1: Um ein Diagramm innerhalb eines Projekts zu löschen, klickt man in dem Entsprechendem Diagramm Auswahlmenü auf das „X“ rechts neben dem zu löschenen Diagramm.

6.2 Code Beispiel

Im Folgenden ist ein Codebeispiel um eine Komponente zu einem Diagramm hinzuzufügen. Wir haben uns dafür entschieden, diese Anleitung zu erstellen, um zukünftigen Teams und Mitarbeitern einen schnelleren Einstieg in die Programmstruktur zu ermöglichen. Als Beispiel wird ein erweiterter Kommentar einem FlowView Diagramm hinzugefügt.

6.2.1 Schritt 1: Model erstellen

Die Klasse *ExtendedCommentComponent* wird im Projekt *FlowDesign.Model* im Ordner Components/FlowView erstellt. Da es sich um eine Komponente handelt, welche spä-

ter einem FlowView Diagramm hinzugefügt werden soll, muss die neue Komponente von *FlowViewComponent* erben.

```
1 // Beispiel 1 - Neues Component Model
2 public class ExtendedCommentComponent : FlowViewComponent
3 {
4     public string Header { get; set; }
5     public string Text { get; set; }
6
7     public ExtendedCommentComponent()
8     {
9         // Type ist in FlowViewComponent als Enum definiert
10        FlowViewType = FlowViewTypes.ExtendedComment;
11    }
12 }
```

Damit im *ExtendedCommentComponent* der richtige Typ der Komponente hinzugefügt werden kann, muss in der Datei Components/FlowView/FlowViewComponent.cs der Enum *FlowViewTypes* um *ExtendedComment* erweitert werden.

```
1 // Beispiel 1 - FlowView Enum in FlowViewComponent
2 public enum FlowViewTypes
3 {
4     Undefined,
5     Modul,
6     InputOutput,
7     // Enum um neuen Typ fuer die neu erstellte Komponente
8     // erweitert
9     ExtendedComment,
```

Damit die Komponente auch gespeichert werden kann, muss in Components/FlowView/FlowViewComponent.cs eine Annotation hinzugefügt werden.

```
1 [XmlInclude(typeof(ModulComponent))]
2 [XmlInclude(typeof(InputOutputComponent))]
3 // Include fuer den Erweiterten Kommentar
4 [XmlInclude(typeof(ExtendedCommentComponent))]
5 public class FlowViewComponent : Component
6 {
7     public FlowViewTypes FlowViewType { get; set; } =
8         FlowViewTypes.Undefined;
9
10    public FlowViewComponent()
```

```
11         Type = ComponentType.FlowView;
12     }
13 }
```

6.2.2 Schritt 2: Anpassung des Component Service

Diese Erweiterung ist im Projekt *FlowDesign.Service* vorzunehmen. Als erstes muss in der Klasse ComponentServices/FlowViewComponentService.cs ein Key hinzugefügt werden.

```
1 // Beispiel 2 - Key in FlowViewComponentService hinzufuegen
2 private const string MODUL_COMPONENT_KEY = "Modul";
3 private const string INPUT_OUTPUT_COMPONENT_KEY =
    "Input/Output";
4 // Neu hinzugefuegter Key
5 private const string EXTENDED_COMMENT_COMPONENT_KEY =
    "Erweiterter Kommentar";
```

Der Key ist der Wert, welcher später in der Auswahlliste der verfügbaren Komponenten im UI angezeigt wird. Er wird als Variable festgelegt, damit bei einer Änderung des Keys nicht an unterschiedlichen Stellen im Code der Wert geändert werden muss.

Damit der Key auch in die Liste der verfügbaren Komponenten auf der UI angezeigt werden kann, muss er einer Liste hinzugefügt werden. Die Methode befindet sich in der gleichen Datei. Die Methode selbst stammt aus dem Interface *IComponentService*, das alle ComponentServices implementiert.

```
1 // Beispiel 2 - Key der Liste "result" hinzufuegen
2 public List<string> GetAvailableComponents()
3 {
4     List<string> result = new List<string>();
5
6     result.Add(MODUL_COMPONENT_KEY);
7     result.Add(INPUT_OUTPUT_COMPONENT_KEY);
8     result.Add(CommonComponentKeys.TEXT_COMPONENT_KEY);
9     result.Add(CommonComponentKeys.COMMENT_COMPONENT_KEY);
10    // Hinzugefuegter Key
11    result.Add(EXTENDED_COMMENT_COMPONENT_KEY);
12
13    return result;
14 }
```

Um dem Diagramm schlussendlich die Komponente hinzuzufügen, wird die Methode *AddComponent(...)* erweitert.

```
1 // Beispiel 2 - Methode um Komponente erweitertern
2 public Component AddComponent(string componentKey, Diagram
diagram, PositionInfo positionInfo)
3 {
4     Component component = null;
5
6     if (componentKey == MODUL_COMPONENT_KEY)
7     {
8         component = new ModulComponent {Name = "Modul"};
9         component.UIProperties.Width = 200;
10    }
11
12    if(componentKey == INPUT_OUTPUT_COMPONENT_KEY)
13    {
14        component = new InputOutputComponent { Name =
15             "Input/Output" };
16    }
17
18    // Neue Komponente
19    if(componentKey == EXTENDED_COMMENT_COMPONENT_KEY)
20    {
21        component = new ExtendedCommentComponent
22        {
23            // Defaultwerte ueber den Objektinitialisierer
24            // festlegen
25            Name = "ExtendedComment",
26            Header = "Erweiterter Kommentar",
27            Text = "Dies ist ein erweiterter Kommentar",
28        };
29    }
30
31    if (component != null)
32    {
33        component.ParentDiagramID = diagram.ID;
34        component.UIProperties.Left =
35            positionInfo.LeftPosition -
36            component.UIProperties.Width * 0.5;
37        component.UIProperties.Top =
38            positionInfo.TopPosition -
39            component.UIProperties.Height * 0.5;
40
41        // Fuegt dem Diagramm die Komponente hinzu
42        diagram.DiagramComponents.Add(component);
43    }
44}
```

```
38
39     return component;
40 }
```

Hier muss noch der Übergabeparameter *componentType* erklärt werden. Die UI kennt bei der Erstellung der Komponenten nur die Werte aus den am Anfang der Klasse festgelegten Keys. Soll nun von der UI aus eine neue Komponente erzeugt werden, übergibt sie der Methode *AddComponent(string componentKey,)* aus *IComponentService* diesen Key. Der ComponentService weiß über diesen Key, welche Komponente erstellt werden soll.

6.2.3 Schritt 3: Erstellung des ViewModels

Diese Änderung ist im Projekt *FlowDesign.UI* vorzunehmen. Zuerst wird dem Ordner *ViewModels/ComponentViewModels/FlowViewViewModels* ein ViewModel mit dem Namen *ExtendedCommentComponentViewModel* hinzugefügt. Die neue Klasse muss von *ComponentViewModel* erben, das die Basisklasse für alle ComponentViewModels ist. Es muss darauf geachtet werden, dass der Namespace *FlowDesign.UI.ViewModels.ComponentViewModels* lautet, da die UI diesen Namespace kennt. Es wäre möglich einen Namespace einzuführen, der *FlowDesign.UI.ViewModels.ComponentViewModels.FlowViewViewModels* heißt und dann nur die ViewModels eines FlowView Diagramms kennt, aber dann muss dieser Namespace in der XMAL in der er verwendet wird definiert werden. Der Einfachheit halber wird hier darauf verzichtet. Hinweis: Visual Studio kann automatisch einen Konstruktor generieren, der zur Basisklasse passt, allerdings wird dann dem *ExtendedCommentComponentViewModel* Konstruktor als ersten Parameter *Component* als Typ übergeben und nicht *ExtendedCommentComponent*. Der Parameter sollte zum richtigen Typ geändert werden, damit später nicht ausversehen eine falsche Komponente übergeben wird.

```
1 // Beispiel 3 - Neues ViewModel erstellen
2 namespace FlowDesign.UI.ViewModels.ComponentViewModels
3 {
4     public class ExtendedCommentComponentViewModel : ComponentViewModel
5     {
6
7         public ExtendedCommentComponentViewModel(ExtendedCommentComponent
8             component, MainPageViewModel parentViewModel) :
9             base(component, parentViewModel)
10    }
11 }
```

Damit die Eigenschaften des erweiterten Kommentars (Header und Text) auch richtig mit der UI verbunden werden können, wird dem *ExtendedCommentComponentViewModel* noch zwei C# Properties hinzugefügt, die als Wrapper für die Properties von *ExtendedCommentComponent* dienen. Dadurch ist es möglich das *PropertyChangedEvent* von ViewModels zu nutzen. Dieses Event informiert die UI, dass sich der Wert einer Property geändert hat und aktualisiert das UI-Element, das mit der Property verbunden ist. Die Wrapper Properties sollten nur erstellt werden, wenn sie auch von der UI aus geändert werden sollen.

```
1 // Beispiel 3 - Anlegen von Wrapper Properties
2 public string Header
3 {
4     get
5     {
6         return ((ExtendedCommentComponent)Component).Header;
7     }
8
9     set
10    {
11        ((ExtendedCommentComponent)Component).Header = value;
12        RaisePropertyChanged();
13    }
14}
15
16 public string Text
17 {
18     get
19     {
20         return ((ExtendedCommentComponent)Component).Text;
21     }
22
23     set
24     {
25         ((ExtendedCommentComponent)Component).Text = value;
26         RaisePropertyChanged();
27     }
28 }
```

Damit die Werte des Erweiterten Kommentars auch im Eigenschaften Panel der UI auftauchen, muss dem *ExtendedCommentComponentViewModel* zwei *SinglePropertyDescriptor* hinzufügt werden. Diese erstellen auf der UI ein Label und ein Textfeld. Die Liste *PropertyDescriptions* ist bestandteil von *ComponentViewModel*. Der erste Parameter einer *SinglePropertyDescriptor* ist der Name der C# Property. Dieser wird genutzt um das DataBinding zu dieser Property herzustellen. Der zweite Parameter ist der Name der auf

der UI als Label angezeigt wird.

```
1 // Beispiel 3 - Hinzufuegen von Eigenschaften
2 public ExtendedCommentComponentViewModel(ExtendedCommentComponent
   component, MainPageViewModel parentViewModel) :
   base(component, parentViewModel)
3 {
4     PropertyDescriptions.Add(new
      SinglePropertyDescription("Header", "Ueberschrift"));
5     PropertyDescriptions.Add(new
      SinglePropertyDescription("Text", "Text"));
6 }
```

Um das ViewModel zu erstellen, wird die Methode *CreateViewModelsFromComponent(...)* der Klasse *ViewModelFactories/FlowViewModelFactory.cs* erweitert. Hier kommt der Typ zum Einsatz der in *FlowViewTypes* definiert wurde. Es wird eine Liste von ViewModels zurückgegeben, da die Möglichkeit besteht, dass ein UI-Element mehrere ViewModels hat. Zum Beispiel besteht eine Linie aus drei ViewModels. Die Linie an sich, und zwei ViewModels für die Anfasser mit denen die Linie verschoben werden kann.

```
1 // Beispiel 3 - Erstellung des
   ExtendedCommentComponentViewModel
2 public List<ComponentViewModel>
   CreateViewModelsFromComponent(Component component,
   Diagram parentDiagram, MainPageViewModel parentViewModel)
3 {
4     FlowViewComponent flowViewComponent =
       (FlowViewComponent)component;
5
6     if(flowViewComponent.FlowViewType == FlowViewTypes.Modul)
7     {
8         return new List<ComponentViewModel> { new
           ModulComponentViewModel((ModulComponent)component,
           parentViewModel) };
9     }
10
11    if(flowViewComponent.FlowViewType ==
   FlowViewTypes.InputOutput)
12    {
13        return new List<ComponentViewModel> { new
           InputOutputComponentViewModel((InputOutputComponent)component,
           parentViewModel) };
14    }
15 }
```

```
16     // ViewModel erstellung fuer den Erweiterten Kommentar
17     if(flowViewComponent.FlowViewType ==
18         FlowViewTypes.ExtendedComment)
19     {
20         return new List<ComponentViewModel> { new
21             ExtendedCommentComponentViewModel((ExtendedCommentComponent)
22                 parentViewModel) };
23     }
24 }
```

6.2.4 Schritt 4: Festlung der Darstellung

Änderungen sind in Pages/MainDiagramPage.xaml vorzunehmen. Im ItemsControl *x:Name="diagram"* muss ein DataTemplate innerhalb des Tags *<ItemsControl.Resources>...</ItemsControl.Resources>* erstellt werden, dass festlegt wie das ViewModel auf der UI dargestellt wird.

```
1 // Beispiel 4 - Darstellung
2 <DataTemplate DataType="{x:Type
3     componentViewModels:ExtendedCommentComponentViewModel}">
4     <componentControls:ComponentContainerControl
5         DataContext="{Binding}"
6             <Border Background="CornflowerBlue"
7                 BorderBrush="Green"
8                 BorderThickness="2"
9                 IsHitTestVisible="False">
10                <StackPanel>
11                    <TextBlock FontWeight="Bold"
12                        Text="{Binding Header}" />
13                    <TextBlock Text="{Binding Text}" />
14                </StackPanel>
15            </Border>
16        </componentControls:ComponentContainerControl>
17    </DataTemplate>
```

Der *componentControls:ComponentContainerControl* sorgt dafür, dass das UI-Element vergrößert/verkleinert und verschoben werden kann.

6.2.5 Ergebnis

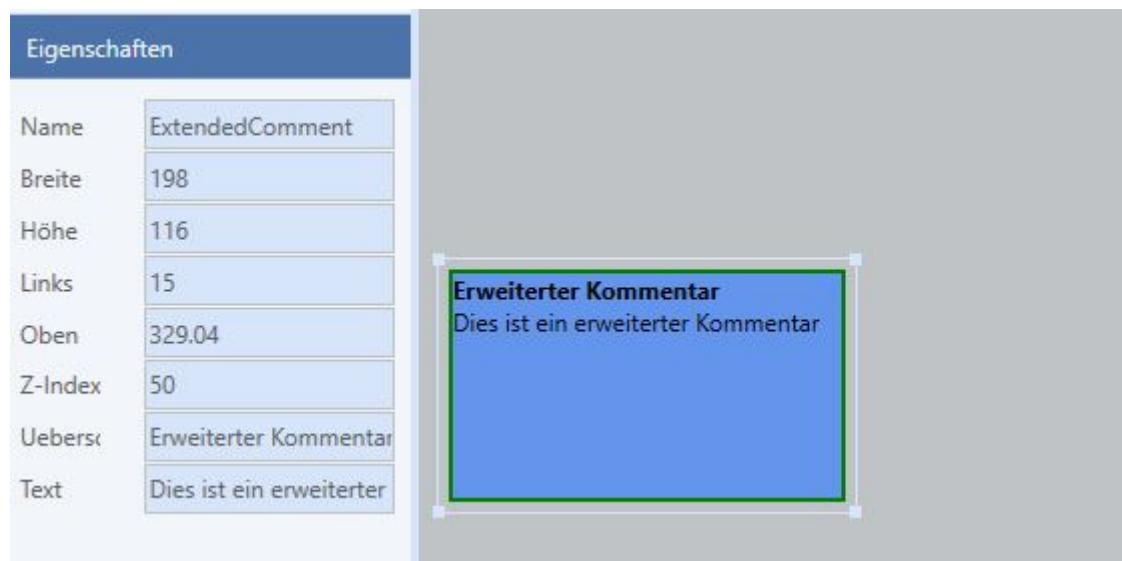


Abbildung 66: Fertige Komponente

6.3 Statistiken

Da die Software einen Prototyp darstellt, welcher von anderen Teams erweitert werden kann, war uns ein gut strukturierter und sauberer Code sehr wichtig. Aus diesem Grund haben wir viel Wert auf gute Programmierung gelegt, was sich in den Metriken widerspiegelt.

In der Abbildung 67 sind allgemeine Werte der Software dargestellt. Um die Metriken zu erstellen wurde das Visual Studio Plugin NDepend⁶ benutzt, welches ein spezielles Tool ist, um .Net Code Metriken zu berechnen.

⁶<http://www.ndepend.com/>



Abbildung 67: NDepend Metriken Allgemein

Während dem Programmieren achteten wir stets auf gute Kommentierung des Codes, wodurch auch in Zukunft die einzelnen Funktionen noch einfach nachzuvollziehen sind. In dem Diagramm (Abbildung 68) sieht man die verschiedenen Abhängigkeiten der einzelnen Module. Wie deutlich erkennbar ist, haben wir uns strikt an die Layered Architecture gehalten, in der die Abhängigkeiten zwischen den Projekten klar geregelt sind. Die verschiedenen Layer sind in dem Kapitel 5.5 (Seite 37) beschrieben.

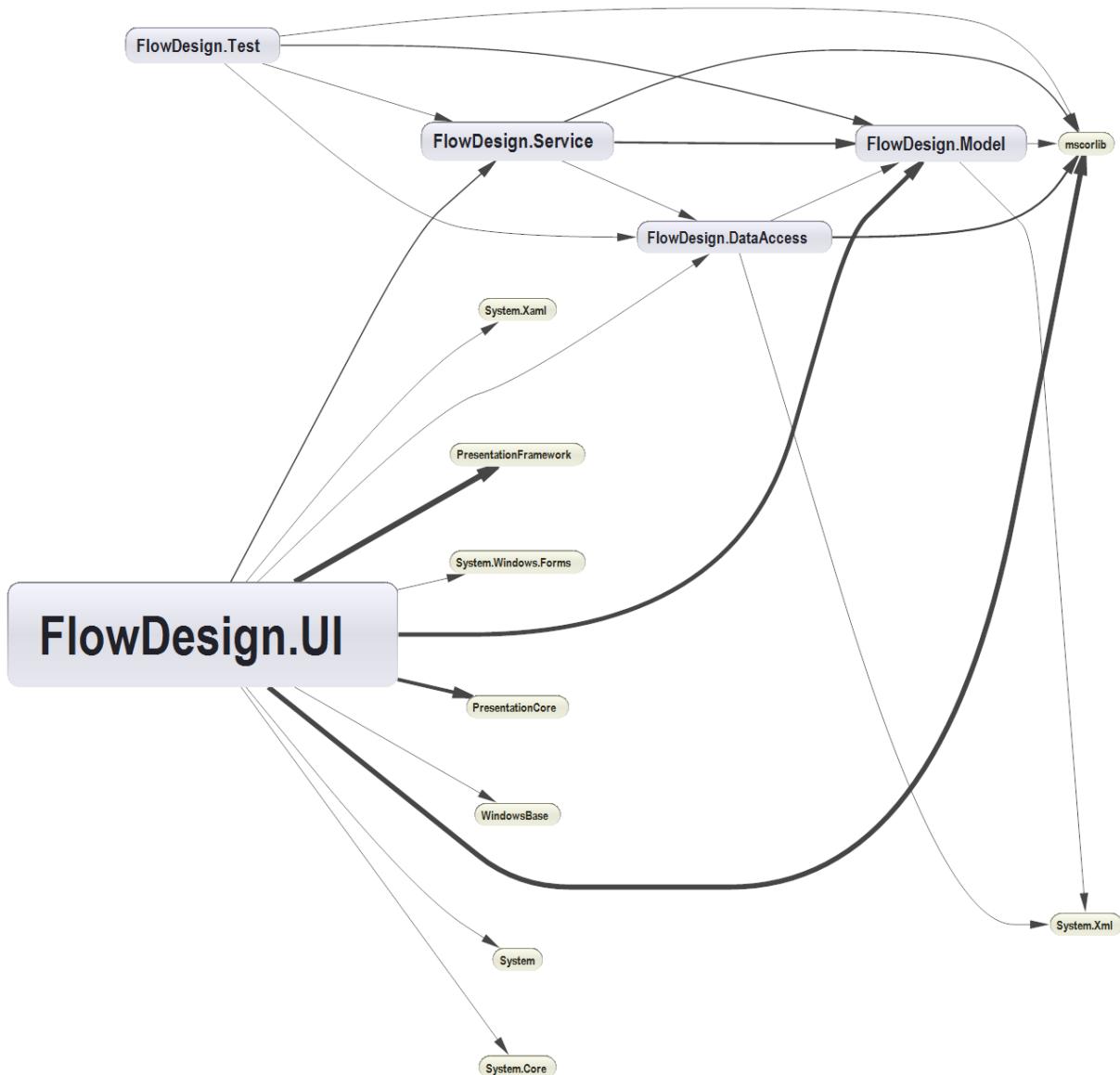


Abbildung 68: NDepend Metriken Abhängigkeiten

Bei der Programmierung haben wir uns stets an die S.O.L.I.D Principles⁷ gehalten. Deutlich wird das in der Abbildung 69. Sie zeigt, welche Module schwierig zu warten sind und welche möglicherweise unbrauchbar sind.

- Abstractness: Wenn ein Modul viele Abstrakte Typen wie Interfaces besitzt, wird es als Abstract dargestellt.
- Stability: Ein Modul wird als Stable dargestellt, wenn die Typen von vielen anderen

⁷https://de.wikipedia.org/wiki/Prinzipien_<u>objektorientierten</u>_Designs#SOLID-Prinzipien

Modulen genutzt werden und eine hohe Abhängigkeiten aufweisen. Somit bedeutet ein hoher Wert bei Instability, dass die Software sehr gut wartbar ist.

Das Model ist in diesem Fall eine Ausnahme, da dieses keine Interfaces besitzen kann und somit als Stable (schlecht wartbar) dargestellt wird.

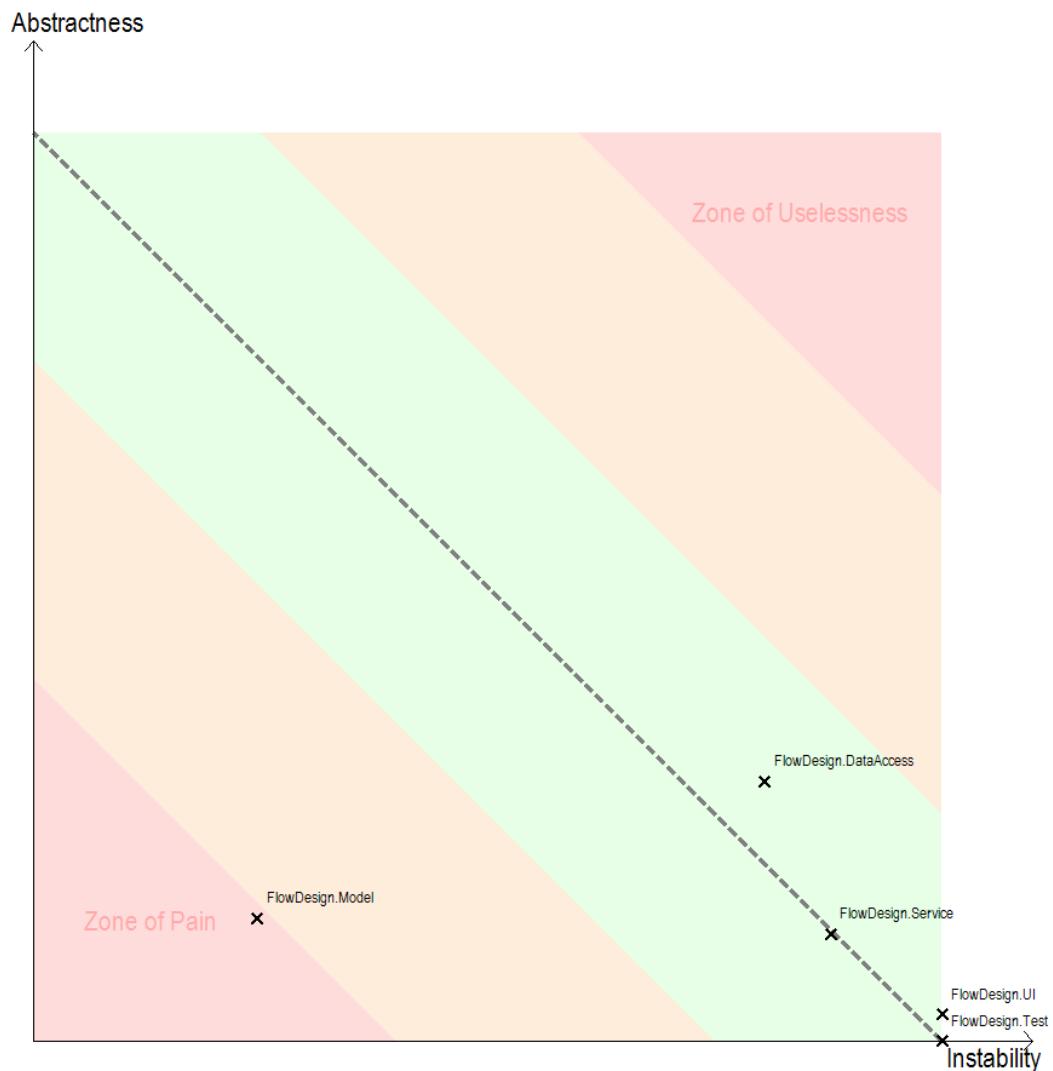


Abbildung 69: NDepend Metriken Abstractness vs. Instability

Weitere Metriken bietet Visual Studio selbst an. Während der Entwicklung nutzten wir diese, um schon während der Implementierung den Code zu testen und somit für die

Zukunft wartbar zu halten. Code Metriken von Visual Studio [1]:

- **Wartbarkeits-Index** – Berechnet einen relativen Indexwert zwischen 0 und 100, der angibt, wie einfach der Code zu verwalten ist. Ein hoher Wert steht für bessere Verwaltbarkeit. Mit farbcodierten Bewertungen können problematische Stellen im Code schnell ermittelt werden. Eine grüne Bewertung liegt zwischen 20 und 100 und gibt an, dass der Code über eine gute Wartbarkeit verfügt. Eine gelbe Bewertung liegt zwischen 10 und 19 und gibt an, dass der Code über eine mäßige Wartbarkeit verfügt. Eine rote Bewertung liegt zwischen 0 und 9 und gibt eine niedrige Wartbarkeit an.
- **Zyklomatische Komplexität** – Misst die strukturelle Komplexität des Codes. Sie wird durch Berechnung der Anzahl unterschiedlicher Codepfade im Programmfluss erstellt. Für ein Programm mit komplexer Ablaufsteuerung sind mehr Komponententests erforderlich, wenn eine gute Codeabdeckung erzielt werden soll, zudem verschlechtert sich die Verwaltbarkeit.
- **Vererbungstiefe** – Gibt die Anzahl der Klassendefinitionen an, die sich bis zum Stamm der Klassenhierarchie erstrecken. Je tiefer die Hierarchie, umso schwieriger ist unter Umständen zu erkennen, wo bestimmte Methoden und Felder definiert oder/und neu definiert werden.
- **Klassenkopplung** – Misst die Kopplung an eindeutige Klassen durch Parameter, lokale Variablen, Rückgabetypen, Methodenaufrufe, generische oder Vorlageninstanziierungen, Basisklassen, Schnittstellenimplementierungen, für externe Typen definierte Felder sowie Attributdekorationen. Gute Softwareentwürfe zeichnen sich durch Typen und Methoden mit hoher Kohäsion und loser Kopplung aus. Eine enge Kopplung deutet auf einen Entwurf hin, der aufgrund zahlreicher gegenseitiger Abhängigkeiten zwischen anderen Typen nur schwer wiederzuverwenden und zu verwalten ist.
- **Codezeilen** – Gibt die ungefähre Anzahl von Zeilen im Code an. Da die Anzahl auf dem IL-Code basiert, entspricht der Wert nicht der exakten Anzahl von Zeilen in der Quellcodedatei. Ein sehr hoher Wert kann darauf hinweisen, dass ein Typ oder eine Methode zu viele Aufgaben ausführt und aufgeteilt werden sollte. Er könnte auch darauf hindeuten, dass der Typ oder die Methode schwierig zu verwalten ist.

In der folgenden Tabelle sind die Visual Studio Code Metriken von FlowDesign dargestellt.

	Wartb.-Index	Zykl. Kompl.	Vererbungst.	Klassenk.	Codez.
Model	93	122	3	24	165
Data Access	89	46	4	25	83
Service	84	81	1	43	183
UI	84	682	10	235	1208

Tabelle 1: Code Metrik Visual Studio

7 Schlusswort

Rückblickend war das Projekt aus unserer Sicht ein voller Erfolg. Es ist nach Projektplan verlaufen und konnte somit fristgerecht abgeschlossen werden. Die gewünschten Features konnten wir implementieren. Für die Zukunft gibt es noch weitere Features und Optimierungen, die von anderen Teams oder Entwicklern implementiert werden können, weshalb der Quellcode unter der Open Source Lizenz zur Verfügung gestellt wird.

Viele der angewandten Vorgehensweisen werden wir aus diesem Grund auch bei den nächsten Projekten beibehalten. Das Zeitmanagement und Meilensteine während der Implementierung werden wir in nächsten Projekten feingranularer festlegen, da wir uns zu Beginn zu sehr auf das allgemeine Diagrammframework konzentrierten und damit etwas Zeit verloren haben, die uns gegen Ende unter Druck gebracht hat. Dennoch sind wir mit dem Ergebnis zufrieden und wünschen zukünftigen Entwicklern viel Erfolg beim Erweitern der Software.

Zum Schluss möchten wir uns für die großartige Zusammenarbeit bei Kevin Erath in diesem Projekt bedanken. Es hat uns nicht nur sehr viel Spaß gemacht, sondern wir konnten auch vieles in der Praxis umsetzen, was wir in Vorlesungen gelernt hatten und sogar einiges Neue dazu lernen. Auch mit dem Flow Design Entwurfskonzept werden wir uns in Zukunft mit Sicherheit noch weiter beschäftigen, da es eine hervorragende Möglichkeit ist, Code sauber, übersichtlich und strukturiert zu halten und Abhängigkeiten zwischen Modulen entgegen zu wirken. Außerdem möchten wir uns bei Professor Reinhard Schmidt bedanken, der uns in diesem Projekt betreut und bei Fragen unterstützt hat.

Literatur

- [1] Microsoft. *Codemetrikwerte*. URL: <https://msdn.microsoft.com/de-de/library/bb385914.aspx>.
- [2] Ralf Westphal. *The Architects Napkin*. Lean Publishing, 2014. ISBN: 978-1495312588.
- [3] Ralf Westphal und Stefan Lieser. *flow-design.org - Fluent software production made easy*. URL: <http://flow-design.org/>.