Projektdokumentation

Autor I – E-Mail
Autor II – E-Mail
HTWK Leipzig

Inhaltsverzeichnis

I. Anforderungsspezifikation

I.1 Initiale Kundenvorgaben

Autor: Jonas Gwozdz

Die Vorgaben unseres Kunden, Prof. Gürtler, ließen uns viele Freiheiten in der Gestaltung des Programms.
Die gegebenen Vorgaben legten das Folgende fest:

- Für die Herstellung eines Werkstücks gibt es in der Regel mehrere Alternativen. Ausgehend von einem Rohteil folgen verschiedene Bearbeitungsschritte, die zum Fertigteil führen. Die einzelnen Bearbeitungsschritte sollen mit einer Bearbeitungszeit und Kosten belegt werden.
- Im Tool soll der entsprechende Graph angelegt werden und Kostenfunktionen sowie Bearbeitungszeiten für die einzelnen Schritte und eine Losgröße eingegeben werden können.
- Kosten fallen pro Bearbeitungsschritt einmalig, pro Zeiteinheit (z.B. Lohn) und pro Werkstück an. AN jeder Maschine werden Rüstzeiten (pro Los) und Bearbeitungszeiten (pro Werkstück) verbraucht.
- Das Tool soll bei Vorgabe all dieser Größen die günstigsten Varianten (nach Kosten, Zeit oder einer Kombination der beiden) in Abhängigkeit von einer vorgegebenen Losgröße berechnen.

I.2 Produktvision

Autor: Alex Hofmann

Product Vision Board:

Target Group	**Needs**	**Product**
-Maschinenbau-Studenten Maschinenbau-Profs -Lehrende	Vgl. zu händisch: einheitlicher, schneller -plattformunabhängig -Open Source -Einfach zu bedienen	-Webanwendung -Als Graph → quasi als Baukasten → Kantengewichtung, Bausteine wählbar -Import/Export von Modellen Normalisierung des Graphen

Die Webanwendung VarG wird entwickelt für Lehrende und Lernende aus dem Maschinenbau Bachelorstudiengang. Diese erleichtert die einheitliche Erstellung, Bearbeitung, Optimierung sowie Im- bzw. Exportierung von sogenannten Variantenfolgegraphen, kurz VarGraphs. Darunter ist eine graphische Übersicht zu verstehen, die die möglichen Varianten eines Produktionsprozesses für ein Werkstück darstellt.

Später überarbeitete Produktvision bzw. neue Projektbeschreibung:
Die plattformunabhängige Open-Source Webanwendung "VarG" soll es Lehrenden und Lernenden aus dem Studiengang Maschinenbau ermöglichen, einfach und schnell Variantenfolgegraphen, kurz VarGraphs, zur Herstellung von Werkstücken zu visualisieren und nach verschiedenen Kriterien die günstigsten Wege berechnen und anzeigen zu lassen. Dafür stehen ihnen viele Features für Aufbau, Funktionsweise, Design, Export und Import zur Verfügung.

I.3 Liste der funktionalen Anforderungen

Autor: Erik Heldt

- Erstellen von Zuständen mit Namen & Kürzel
- Erstellen von Arbeitsschritten mit Namen & Kürzel zwischen je 2 Zuständen
- Zuweisen von (Rüst-)Zeitkosten, (Rüst-)Geldkosten & Losgröße zu Arbeitsschritt
- Anzeigen des günstigsten Weges im Graph, berechnet nach der angegebenen Kostenart
- Lokaler Export als Bilddatei oder importierbarer JSON & Lokaler Import als JSON
- Hochladen in online gehostete Datenbank & Laden aus online gehosteter Datenbank
- Login-Management für Zugriffskontrolle auf Anwendung
- Rollen-Management (Student, Professor) für Zugriffsrechte auf Datenbank

I.4 Liste der nicht-funktionalen Anforderungen

Autor: Erik Heldt

- Schnelle Einarbeitung in die Anwendungsumgebung
- Einfacher & intuitiver Umgang mit den Programmkomponenten und -funktionen
- Stabiler & konsistenter Programmablauf, keine Abstürze oder Verluste von Dateien

- Kompatibilität mit so vielen modernen Browsern wie möglich
- Sicherheit & korrekte Funktionalität des Login-Algorithmus und des DB-Rollenmanagements
- Datenschutz bei Login-Sessions einhalten

I.5 Weitere Zuarbeiten zum Produktvisions-Workshop

Autor: Erik Heldt

Für den Produktvisions-Workshop wurden 4 Dokumente erstellt, welche unterschiedliche Aspekte des Anwendungsentwurfs behandeln.

Die erste Ausarbeitung zeigt Ideen zur Darstellung der GUI inklusive eines interaktiven GUI-Prototyps auf Adobe XD, die Zweite ist ein Epic bzw. eine Zusammenfassung vieler User-Stories zu allgemeinen Anforderungen an die Funktionalität. Das dritte Dokument geht genauer auf spezifische Kernfunktionen ein und das vierte umfasst die Datenmodellierung des Programms.

Die nachfolgende Liste an Zuarbeiten sind klickbare Verweise auf die jeweiligen Dokumente im "zuarbeiten"-Ordner.

I.5.1 Zuarbeit von Linus Herterich, Jonas Gwozdz, Julius Hohlfeld

VarG GUI

I.5.2 Zuarbeit von Erik Heldt, Alaa Aldin Karkoutli

Erstes Epic

I.5.3 Zuarbeit von Lennart Buchmann, Nils Buxel, Matthias Berger

Kernfunktionalität

I.5.4 Zuarbeit von Tim Henning, David Koch

Datenmodell

I.6 Liste der Kundengespräche mit Ergebnissen

Autor: xxx

XXX

II. Architektur und Entwurf

II.1 Zuarbeiten der Teammitglieder

Autor: Erik Heldt

Für die Technologierecherche informierte sich das Team über verschiedene Technologien, mit denen die Anwendung entwickelt werden kann. Außerdem fassten wir erste Ideen zur Graphenanordnung zusammen und legten Coding Conventions fest. Die Ausarbeitungen wurden in den nachfolgenden Dokumenten festgehalten.

Die nachfolgende Liste an Zuarbeiten sind klickbare Verweise auf die jeweiligen Dokumente im "zuarbeiten"-Ordner.

II.1.1 Zuarbeit von Tim Henning

Django und Python

II.1.2 Zuarbeit von Erik Heldt

Ruby on Rails
Ruby on Rails (kurz)
Graphenanordnung

II.1.3 Zuarbeit von David Koch

Java Canvas
Datenbanken

II.1.4 Zuarbeit von Matthias Berger, Nils Buxel

Coding Guidelines

II.1.5 Zuarbeit von Nils Buxel

Coding Conventions CSS

II.1.6 Zuarbeit von Julius Hohlfeld

Angular

II.1.7 Zuarbeit von Lennart Buchmann, Alaa Aldin Karkoutli, Jonas Gwozdz, Linus Herterich

Bibliotheken zur Graphenerstellung

II.2 Entscheidungen des Technologieworkshops

Autor: Erik Heldt

Nach ausgiebigen Recherchen über verschiedenste Programmiersprachen, Frameworks und Bibliotheken entschieden wir uns für eine Webanwendung auf Basis von HTML/CSS/JavaScript.

Wir haben uns weiterhin auf das JS-Framework Vue.js geeinigt, da es viele Vorteile für die Front-End-Entwicklung mit sich bringt und von den vielen untersuchten Frameworks am intuitivsten erschien. Außerdem haben wir nach einer JS-Bibliothek zur Graphdarstellung recherchiert und unter verschiedenen Kandidaten stach Cytoscape mit seinen vielen Funktionen zur Graphenerstellung und -editierung am meisten heraus, was wir somit auch in unsere Architektur integrierten.

Bei der Programmierumgebung waren wir uns schnell einig, dass Visual Studio Code am besten für unsere Ansprüche geeignet ist. Wir installierten die IDE zusammen mit dem Plugin ESLint zur Unterstützung der Einhaltung standardmäßiger Coding Conventions.

II.3 Überblick über Architektur

Autor: Linus Herterich

VarG ist eine Web-App nach dem Client-Server Modell, wobei der Großteil der Berechnungen per JavaScript auf dem clientseitigen Browser durchgeführt werden.
Serverseitig wird eine Datenbank (inkl. API-Schnittstelle) zum persistenten Speichern der erstellten Graphen angeboten.

Die Architektur der Web-App basiert auf dem JavaScript-Webframework "Vue.js", mit dem Webanwendungen nach dem MVVM Muster (Model View ViewModel) realisiert werden können. Die gesamte App ist nach logischen Sites (Seiten, bei denen sich die URL ändert) und Components (wiederverwendbare, abgeschlossene Software-Schnipsel) aufgebaut. Jede Vue Component (.vue Dateien) enthält ein HTML-Template (GUI), sowie Daten, mit denen das Template befüllt wird. Zudem werden Funktionen definiert, die entweder zu bestimmten Laufzeitbedingungen der App oder durch Events und Trigger aufgerufen werden. Die Kommunikation zwischen Components wird über Vererbungen zu Eltern-/ Kind-Components realisiert.
Die Web-App besteht im Entwicklungszustand aus vielen hunderten Dateien, welche vom Framework verwaltet werden. Sobald die App in den Produktionsstatus wechselt, muss das Projekt kompiliert werden. Dies übernimmt ebenfalls das Framework, welches hierfür Technologien wie "WebPack" einsetzt. So bleiben lediglich wenige HTML, CSS und JavaScript Dateien übrig, die anschließend auf einem Web-Server (z.B. Apache) zur Verfügung gestellt werden müssen.

Um die Darstellung einheitlich zu halten, haben wir die UI-Bibliothek 'Vuetify" genutzt. Diese hält sich an den Industriestandard "Material Design" von Google. Damit konnten wir alle unsere im Vorfeld erstellten Design-Konzepte umsetzen. Um an den "Vuetify" Elementen weitere optische Anpassungen vorzunehmen haben wir die CSS-Language-Extension "less" verwendet. Mit dieser ist es möglich, übersichtliche und einheitliche Style-Vorgaben auf die Design-Komponenten anzuwenden.

Damit alle Daten komponentenübergreifend auf einen gemeinsamen Datenstamm zugreifen können und die Daten auch nach einer Session persistent gespeichert werden können, haben wir die Vue.js-Erweiterung "Vuex" eingesetzt. Diese bietet eine zentralisierte Speichermöglichkeit für alle Daten, die übergreifend verwendet werden müssen (beispielsweise Log-In Daten oder der Zustand des VarGraphs).

Für die Darstellung des Graphen (Knoten + Kanten und deren Beschriftung) haben wir die JavaScript Bibliothek "Cytoscape.js" verwendet. Die Bibliothek hält alle Graph-Daten in einem JavaScript Objekt, auf das mit verschiedenen API-Funktionen zugegriffen werden kann. Die Darstellung des Graphen wird über ein Canvas HTML Element realisiert, in welches Cytoscape die angelegten Knoten und Kanten zeichnet. Cytoscape bietet ebenfalls eine Hand voll Algorithmen zur analytischen Auswertung des Graphen. Da die Optimierung des VarGraphs allerdings zusätzlichen Bedingungen und Parametern unterliegt, wurde ein eigener VarGraph-Optimierungsalgorithmus entwickelt.

Bei der Wahl der serverseitigen Architektur haben wir eine REST-konforme (Representational State Transfer) Architektur eingesetzt, an dessen Ende eine MySQL Datenbank zur Speicherung der Cytoscape Objekte, sowie Authentifizierungsdaten steht. Auf die Daten der Datenbank greift eine API-Schnittstelle zu, welche mit Node.js umgesetzt ist (weitere Details zur Schnittstelle: siehe II.4 - Schnittstellen). Anfragen an die API werden mit dem "axios" Framework per "Promise-based" HTTP-Requests gestellt. Die HTTP-Requests folgen einem klaren Schema, welches vom serverseitigen Node.js interpretiert und an die Datenbank weitergeleitet wird.

Um die Web-App großflächig zu testen haben wir uns zum einen für das Framework "Cypress" entschieden, welches Integrationstests anhand der HTML-Elemente übernimmt. Cypress wertet aus, ob bestimmte Elemente unter bestimmten Bedingungen vorhanden sind beziehungsweise spezielle Eigenschaften aufweisen. Die Cypress Tests haben wir auch erfolgreich an die "CI / CD Pipeline" von GitLab angeschlossen, sodass nach jedem push die Tests durchlaufen (Stichwort: Regressionstest).

Zum anderen haben wir das Framework "jest" für Unit-Tests eingesetzt, mit dem einzelne Funktionen auf ihre Richtigkeit überprüft werden können. Vor allem für die Optimierungsalgorithmen sind isolierte Tests nötig gewesen.

Um eine Client-Server Architektur zu simulieren haben wir "Docker" eingesetzt. Dieses Tool erlaubt es, virtuelle Maschinen zu erstellen, welche untereinander kommunizieren können. Für Entwicklungszwecke haben wir einen Docker-Container für eine MySQL Datenbank und einen Node.js-Webserver (API Schnittstelle) erzeugt. Ein weiterer Docker-Container wurde eingesetzt, auf dem "Adminer" läuft. Mit diesem Tool ist es möglich, die MySQL-Datenbank komfortabel anzuzeigen und SQL-Zugriffe auszuführen.

II.4 Definierte Schnittstellen

Autor: Julius Hohlfeld

VarGs Funktionalitäten erforden eine Datenbank um die erstellten Graphen speichern und wieder abrufen zu können.
Um den Zugriff auf die Datenbank zu kontrollieren benötigen wir eine definierte Schnittstelle (bzw. API) zwischen Client, Webserver und Datenbank.

Diese Schnittstelle ist RESTfull - d.h. sie folgt einigen der sog. REST-Constraints. Eine Übersicht inwiefern zu REST und welche Bedeutung es für das Projekt hat, findet sich im GitLab Wiki unter 'API Dokumentation'.
Die Schnittstelle setzt sich wie folgt zusammen:

- **Vue**
  Framework für Client + Axios Modul für asnychrone (promise-based) HTTP-Requests
- **Express**
  Serverseitiges Node-Module für Webserver: hört angemeldete Ports auf Requests ab, die dem URI-Modell entsprechen
- **Node.js**
  Serverseitige Programmierung des Webservers mit mysqljs als Driver, um auf die Datenbank zuzugreifen
- **DB**
  MySQL-Datenbank auf extra Server

Diese Struktur (kurz VenDB) entspricht einer Anpassung des sog. MEAN-Stacks auf das VarG-Projekt (MongoDB, Express, Angular, Node.js).
Dabei erfolgt jeglicher Austausch der Graphdaten im JSON Format damit auf die cytoscape.js Funktion zum Laden des Graphen zugegriffen werden kann.

II.4.1 Client

Der Client enthält Trigger durch Events, welche Requests an den Webserver senden. Z.B.: das Aufrufen des Datenbankfenster löst eine Anfrage aus, welche alle Graphen des aktuellen Nutzers abfragt. Diese werden durch das Axios-Modul umgesetzt. Nachdem der Trigger ausgelöst wird, schickt der Client eine asynchrone Request. Diese wird vom Webserver verarbeitet, welcher dann eine Antwort schickt. Diese kann von Axios aufgefangen werden (axios."request"(url,).then(response =>).catch(error =>)).

II.4.2 Server

Der durch Express und mysqljs programmierte Webserver definiert folgende mögliche Zugriffstellen auf die Datenbank:

- **Get-Requests**
  - **graph**
    Fragt alle Graphen aus der Datenbank ab - für Admin reserviert.
  - **graph/:id?**
    Fragt einen spezifischen Graphen (entsprechend der ID) ab.
  - **graph/meta**
    Fragt Metadaten z.B.: Namen, Id, Stückzahl usw. ab für die Graphen des Nutzers ab.
- **Put-Requests**
  - **graph/:id?**
    Client schickt Server eine Repräsentation des Graphen in Json um einen bereits existierenden Graphen (entsprechend der ID) zu überschreiben.
- **Post-Requests**
  - **graph?**
    Client schickt Server eine Repräsentation des Graphen in Json um einen neuen Eintrag für den Nutzer zu erzeugen.
- **Delete-Request**
  - **graph/:id?**
    Spezifizierter Graph (entprechend der ID) wird aus der Datenbank gelöscht.

Das '?' bedeutet, dass hier auf bestimmte URL Queries geachetet werden kann. Das ist nützlich um z.B.: einen Nutzer nur auf seine eigenen Graphen zugreifen zu lassen. Diese werden dann in die entsprechenden Queries umgewandelt.

II.5 Liste der Architekturentscheidungen

Autor: xxx

XXX (bewusste und unbewusste Entscheidungen mit zeitlicher Einordnung)

III. Prozess- und Implementationsvorgaben

III.1 Definition of Done

Autor: Tim Henning

Im Allgemeinen wurde in dem Projekt die Definition von "doneness" nicht all zu umfangreich gestaltet, da es für viele Teammitglieder eines der ersten Softwareprojekte war. So wurden als Defintion of Done folgende Punkte für alle Userstories aufgestellt:

- >50% Testabdeckung
- Technische Kommentare im Code
- Einhaltung der festgelegten Code Konventionen

Das Team hatte an sich zu den meisten Zeitpunkten eine klare Vorstellung was einen "fertigen Entwurf" kennzeichnet und wurde so auch in den Reviews untereinander kommuniziert. Dies wiederum führte zu einer klaren Transparenz im Team, was die Qualität des Produktes erhöhte und das Zusammenarbeiten erleichterte. Größtenteils wurde sich an die allgemeinen Akzeptanzkritieren gehalten und viele Backlog-Einträge als "done" erklärt. Zu fast jeder Komponente wurde getestet und zu den Methoden der einzelnen Komponenten wurden erklärende sinnvolle Codekommentare geschrieben. Außerdem wurde im Team umfangreich kommuniziert und die Kriterien angepasst, wenn die Fertigstellung einer Userstorie doch mal nicht gänzlich klar war. So wurde es ermöglicht nach der Hälfte des Projektes, am Ende jedes Sprints einen fertigen Productionbuild dem Kunden zu liefern.

III.2 Coding Style

Autor: Jonas Gwozdz

Beim Schreiben unseres Programmcodes haben wir uns an folgende Coding Conventions gehalten.

- Zeilenlänge: maximal 80 Zeichen
- Kommentare und Dokumentation
  - Kommentare auf Englisch
  - Klassen und Methoden in kurzen, prägnanten Sätzen beschreiben
  - Unnötige Kommentare vermeiden
  - Kommentare aktuell halten
- Einrückung und Zeilenumbrüche
  - 2 Leerzeichen statt Tabulator
  - ,{ ‘ hinter Methodendeklaration
  - ,} ‘ in neuer Zeile auf gleiche Einrückungsebene
  - Optionale Zeilenumbrüche für Übersichtlichkeit
  - Nur ein Import pro Zeile
- Leerzeichen
  - Vor und nach binären Operationen
    - Ausnahme nur im Fall von Verdeutlichung unterschiedlicher Prioritäten
  - Keine Leerzeichen vor und nach Klammern
  - Keine Leerzeichen vor Kommata und Semikolon
  - Leerzeichen nach Kommata
  - Keine Leerzeichen am Zeilenende
- Konsistentes Benennungsschema
  - Deskriptive Namen verwenden
  - mixedCase für Variablen
  - GROßSCHREIBUNG für Konstanten

  - Keine Umlaute
  - Reservierte Schlüsselwörter beachten
  - Immer auf Englisch
  - Bezeichner von Booleanwerte sollen Zustand beschreiben, der wahr oder falsch sein kann
  - Hilfsvariablen möglichst gleich benennen
  - Übergabe von Attributen an Konstruktoren
    * ‚length' als Attribut, ‚_length' als Argument

- Textcodierung UTF-8

Best Practice

- Allgemeines
  - Kein ‚language' Tag verwenden
  - Wiederholungen Vermeiden
  - Dopplungen vermeiden
- Variablen und Objekte
  - Keine globalen Variablen
  - Lokale Variablen, auch Zahlenvariablen zu Beginn deklarieren und initialisieren
  - Deklaration mehrerer Variablen können zusammengefasst werden
  - Datentyp wird über die Initialisierung zugewiesen
  - Kapselung mittels Namespace
  - Keine Deklaration mittels ‚new ... ()'
- Funktionen
  - Vergleiche mittels ‚===‘
  - Unter keinen Umständen ‚eval()' benutzen
  - Keine ‚with' Statements
  - Keine ‚for (... in ...)' Loops
  - Jeder ‚switch' hate einen ‚default' Case
  - Vorsicht bei Verwendung von ‚typeof()'
  - Nicht erhaltene Argumente gelten als ‚undefined'

III.3 Zu nutzende Werkzeuge

Autor: Linus Herterich

Im Folgenden werden die Werkzeuge erwähnt, mit denen wir die Software entwickelt haben. Zudem wird darauf eingegangen, über welche Kanäle kommuniziert wurde.

III.3.1 Voraussetzungen

Das Versionsmanagement-Tool "GitLab" sowie das Zeitmanagement-Tool "YouTrack" wurden zu Beginn des Projekts vorgeschrieben. Die Commits in "GitLab" werden jeweils mit der ID des zugehörigen YouTrack-Tickets am Anfang des Commit-Titels versehen.
Damit das gesamte Team einheitliche Versionen der verwendeten Bibliotheken benutzt, wird der Paketmanager "npm" verwendet. Mir diesem lassen sich Pakete (und deren Versionen) definieren, welche für das Projekt benötigt werden.
Damit am Projekt gearbeitet werden kann, muss sich somit jedes Teammitglied die LTS- Version von Node.js (welches npm enthält) installieren.
Sobald Node.js global installiert ist, kann im "code" Verzeichnis der Befehl "npm install" ausgeführt werden, um die benötigten Bibliotheken zu installieren.

III.3.2 Compiler

Achtung: Das Kompilieren funktioniert erst, sobald die Bibliotheken mit dem Befehl "npm install" (im /code Verzeichnis) installiert wurden.
Um Änderungen des Projektes einzusehen, muss das Projekt kompiliert werden. "Vue.js" bringt bereits einen Echtzeit-Compiler mit, welcher reagiert, sobald Änderungen an Dateien im "code" Verzeichnis gemacht wurden. Um diesen Compiler aufzurufen, muss der npm-Befehl "npm run serve" im "code" Verzeichnis aufgerufen werden.
Um das Projekt nicht während der Entwicklung zu kompilieren, sondern für die Produktion freizugeben, muss der Befehl "npm run build" im "code" Verzeichnis aufgerufen werden. Es werden anschließend die kompilierten Dateien im Verzeichnis "code/dist" abgelegt. Diese können anschließend auf einem Webserver (z.B. Apache HTTP Server) hochgeladen werden.

III.3.3 Entwicklungsumgebung

Für die Entwicklung der Software wird der freie Quelltext-Editor "Visual Studio Code" von Microsoft verwendet. Dieser ist plattformunabhängig und kann durch zahlreiche Erweiterungen angepasst werden. Beispielsweise kann durch das Plugin "Vetur" die Vue.js-eigene Syntax vervollständigt und hervorgehoben werden.
Weitere Einstellungsvorgaben bezüglich der Entwicklungsumgebung wurden nicht getroffen. Es muss allerdings darauf geachtet werden, dass die Coding-Conventions durch automatische Formatierungen eingehalten werden.

III.3.4 CI / CD Pipeline

In der CI / CD Pipeline unseres Versionsmanagement-Tools, die nach jedem Git-Push ausgeführt wird, werden folgende Operationen durchgeführt:

- Test, ob das Projekt kompiliert (inklusive Syntaxprüfung durch ES-Lint)
- Cypress Tests durchführen (siehe "Überblick über Architektur")
- LaTeX Doku kompilieren

Sollte einer der Punkte fehlschlagen, wird der Autor des Git-Push's per E-Mail darüber informiert. Somit ist die Wahrscheinlichkeit, dass bestehende Features durch neue Entwicklungen längerfristig "zerstört" werden, möglichst gering.

III.3.5 Docker

Um die Client-Server Architektur des Projektes lokal zu simulieren, wird die Container-Virtualisierungssoftware "Docker" verwendet. Mit dieser haben wir einen Webserver simuliert, auf dem die Datenbank ausgeführt und verwaltet wird (siehe "Überblick über Architektur"). Die Container werden im Projekt-Ordner "docker" definiert.

III.3.6 Kommunikationstools

Zu Beginn des Projekts wurde sich auf das kostenlose Kommunikationstool "Slack" geeinigt. Mit diesem ist es möglich, in verschiedenen Kanälen Text, Dateien und Medien auszutauschen. Auch private Konversationen, sowie Kleingruppen-Chaträume sind in diesem Tool möglich. Die Software kann sowohl als App installiert, als auch im Browser verwendet werden.
Da wir über die Weihnachtsferien einen Sprint durchgeführt haben, führten wir Mitte Dezember das Tool "Discord" ein, mit dem es möglich ist, sich in Echtzeit-Sprachchats zusammenzufinden. Dazu ist es möglich, seinen Desktop zu teilen, womit sich das Tool bestens eignet, um räumlich getrennt über Code-Passagen oder neue Features zu sprechen.
Die Kombination beider Tools hat problemlos funktioniert und uns auch während des Lockdowns in der "Corona-Krise" geholfen. Da wir die Tools bereits frühzeitig eingesetzt haben, war kaum eine Um- bzw. Eingewöhnungszeit zu Beginn der präsenzfreien Zeit notwendig.

IV. Sprint 1

IV.1 Ziel des Sprints

Autor: Erik Heldt

Der erste Sprint des VarG-Projekts lief vom 05.12.2019 bis zum 16.12.2019. Ziel war es, eine fundamentale Struktur und grundlegende Funktionalitäten für die Anwendung zu entwickeln, auf denen man später weiter aufbauen kann. Währenddessen konnte man allgemeine Erfahrungen mit dem Ablauf eines Sprints machen.

IV.2 User-Stories des Sprint-Backlogs

Autor: Erik Heldt

Grundstruktur Die Anwendung sollte zu Beginn ein grundlegendes Fundament aufweisen, damit sich alle Teammitglieder vorstellen können, wie am Ende das Programm aussehen soll. Dazu gehörte zu Beginn das Design der Startseite mit dem VarGraph im Zentrum und der Einbindung von Cytoscape in die Programmstruktur.

Datenstruktur für Knoten Es sollte mit Hilfe von Cytoscape herausgefunden werden, wie man Knoten im Programmcode hinzufügen und speichern kann. Dafür sollte dann eine Datei im Programm angelegt werden.

Knoten zu bestehender Datenstruktur hinzufügen Die Anwendung sollte eine einfache Funktionalität zum Erstellen neuer Knoten aka Produktionsschritte erhalten, um sich mit den Cytoscape-Funktionen näher vertraut zu machen. Hier war erstmal noch keine graphische Darstellung in der GUI notwendig, es reichte per Console logs zu testen.

Darstellung eines Graphen in Weboberfläche In der Anwendung sollte zunächst ein statischer Graph mit Hilfe einer Cytoscape-Datenstruktur sichtbar dargestellt werden, damit man sehen konnte, wie so ein „CytoGraph“ überhaupt aussieht. User-Interaktion war hier noch nicht notwendig.

Kanten anlegen Zusätzlich zu Knoten sollten auch Kanten zwischen bestehenden Knoten hinzugefügt werden können. Diese Kanten sollten mit verschiedenen Attributen in der Cytoscape-Datenstruktur gespeichert werden.

Berechnung verschiedener Eigenschaften Anhand der mit den Kanten gespeicherten Attribute sollte eine Funktionalität entwickelt werden, welche die Gesamtkosten (Auswahl von Geld oder Zeit) aller unterschiedlichen Pfade berechnen und anzeigen sollte. Dies war der erste Schritt in Richtung Optimierung, d.h. später sollte diese Funktionalität automatisch den günstigsten Pfad herausfinden und anzeigen.

IV.3 Liste der durchgeführten Meetings

Autor: Erik Heldt

- Planning - 05.12.2019
- Weekly Scrum 1 - 09.12.2019
- Weekly Scrum 2 - 12.12.2019
- Review - 16.12.2019
- Retrospektive - 19.12.2019

IV.4 Ergebnisse des Planning-Meetings

Autor: Erik Heldt

Im Planning-Meeting erklärten die Projektmanager zu Beginn noch einmal kurz, wie ein Sprint im Allgemeinen abläuft und haben auf die Bedeutsamkeit der Coding Guidelines hingewiesen. Anschließend wurden die ersten User-Stories vom Project Owner vorgestellt und von den Bachelorstudenten per Finger-System in ihrer Komplexität eingeschätzt. Weiterhin wurde festgelegt, dass die Bachelorstudenten während des Sprints die User-Stories selbst in Tasks aufteilen und diese dann bearbeiten sollen.

IV.5 Aufgewendete Arbeitszeit pro Person+Arbeitspaket

Autor: Erik Heldt

Arbeitspaket	Person	Start	Ende	h	Artefakt
Vue.js "Getting Started" Tutorial durcharbeiten (für alle)	Buchmann, Lennart	07.12.19	07.12.19	3	Tutorial abgeschlossen
Beispielgraph erstellen	Buxel, Nils	09.12.19	09.12.19	1	index.js
Kürzesten Weg mit A*-Algorithm berechnen u anzeigen lassen	Buxel, Nils	16.12.19	16.12.19	1	index.js
Funktionen zu Buttons hinzufügen	Gwozdz, Jonas	14.12.19	16.12.19	4	MenuControls.vue
Task: Einbindung in Vue-Dateistruktur	Heldt, Erik	15.12.19	15.12.19	3	MenuControls.vue, BasicData.js
Graphenanordnung	Heldt, Erik	05.12.19	05.12.19	3	Graphenanordnung.pdf
Vue.js "Getting Started" Tutorial durcharbeiten (für alle)	Heldt, Erik	11.12.19	11.12.19	2	Tutorial abgeschlossen
Funktionen zu Buttons hinzufügen	Henning, Tim	10.12.19	10.12.19	2	MenuControls.vue
Vue.js "Getting Started" Tutorial durcharbeiten (für alle)	Henning, Tim	06.12.19	06.12.19	3	Tutorial abgeschlossen
Einbindung von Cytoscape in Vue	Herterich, Linus	10.12.19	10.12.19	4	index.js
Buttons für Knoten und Kantenerstellung	Herterich, Linus	13.12.19	13.12.19	3	CreateControls.vue
Knoten zu Graph hinzufügen	Herterich, Linus	16.12.19	16.12.19	2,5	index.js, CreateControls.vue
Grundstruktur aufbauen	Herterich, Linus	05.12.19	07.12.19	9,5	Vue-Dateistruktur, sämtliche Startkom
Task: Basic Datenstruktur	Hohlfeld, Julius	15.12.19	15.12.19	8	BasicData.js, MenuControls.vue

IV.6 Konkrete Code-Qualität im Sprint

Autor: Erik Heldt

Zu Beginn wurde viel experimentiert und hauptsächlich sollte der Code erstmal ein funktionierendes Programm erzeugen, weswegen weniger auf die Qualität geachtet wurde. Trotzdem wurde sich größtenteils an die Coding Conventions gehalten und bereits einige Kommentare verfasst.

IV.7 Konkrete Test-Überdeckung im Sprint

Autor: Erik Heldt

Da der erste Sprint größtenteils nur zur Erstellung einer grundlegenden Datenstruktur und zur Einarbeitung in JavaScript und den genutzten Frameworks bzw. Bibliotheken gedient hat, gab es noch keine Tests.

IV.8 Ergebnisse des Reviews

Autor: Erik Heldt

Im ersten Review-Meeting stellten die Bachelorstudenten ihre Ergebnisse aus dem Sprint vor und die Manager gaben ihr Feedback dazu. Da sich die meisten Teammitglieder noch nicht richtig in Vue.js und Cytoscape einarbeiten konnten und teilweise große Schwierigkeiten mit den Frameworks hatten, gab es noch viele offene Aufgaben und nicht jeder hatte etwas vorzuzeigen.

Als erstes stellten Julius H. und Erik die Datenstruktur für die Knoten vor. Weiterhin zeigte Julius, wie ein Knoten in der Anwendung dargestellt wird und dass dieser durch ungeschickte Verschiebung und Skalierung aus der GUI verschwinden kann. Deshalb kamen Vorschläge, zukünftig den Zoom zu limitieren und das grundsätzliche Graph-Layout nochmal zu überarbeiten.

Um allen den Einstieg in die neuen Programmiersprachen und Bibliotheken etwas zu vereinfachen, stellte daraufhin Linus die Grundstruktur vor und erklärte noch einmal genau die einzelnen Elemente in der Dateistruktur. Weiterhin zeigte er, wie man ESLint-Fehler bei der Konsolenausgabe verhindern kann.

Danach wurde zwischen den Managern und den Bachelorstudenten noch die zukünftige Berechnung der kürzesten Wege und die unbearbeiteten User-Stories besprochen und dass diese in den nächsten Sprint mit einfließen werden.

Zum Schluss wurden noch ein paar allgemeine Fragen zum Testen und zu Git geklärt.

IV.9 Ergebnisse der Retrospektive

Autor: Erik Heldt

In der Retrospektive konnte jedes Teammitglied vor an die Tafel gehen und verschiedene Aspekte des Sprints mit einem Strich in einer Tabelle bewerten.

Die Bewertung ging ausgeglichen aus. Die Gruppenleistung und das Gesamtergebnis waren gut, aber die Einzelleistungen der meisten Teammitglieder nicht. Viele Aufgaben blieben offen und wurden nicht erledigt, wozu in der Diskussion verschiedene Gründe angeführt wurden. Einerseits war es für die meisten schwer, sich selbst in die neue Programmierumgebung samt den Frameworks und Bibliotheken einzuarbeiten. Andererseits wussten viele nicht, was und wie viel sie machen sollten, was auf die nicht festgelegte Aufgabenzuteilung im Planning und die schlechte Kommunikation im Team während des Sprints zurückgeführt wurde. Letzteres Problem plante man damit zu lösen, in zukünftigen Plannings immer direkt Verantwortliche für bestimmte User-Stories festzulegen und entsprechende Tickets sofort im Anschluss zu erstellen und zuzuweisen.

Beim Thema der Daily Meetings ist man zu dem Schluss gekommen, dass diese wenn möglich immer persönlich bleiben sollten und nur in Ausnahmefällen online z.B. über Discord stattfinden sollten. Weiterhin wurde diskutiert, ob die Zeitspanne zwischen Donnerstag und Montag evtl. zu kurz ist, um schon weitreichende Ergebnisse zu erzielen, da am Wochenende einige Teammitglieder nicht programmieren können. Deshalb sollten die ersten Meetings beim nächsten Sprint stattdessen Montag und Donnerstag stattfinden.

Ein weiterer Themenpunkt war die Organisation im Git. Es wurde festgelegt, dass der Master-Branch während des Sprints unberührt bleiben sollte, da dieser immer lauffähig sein muss. Stattdessen sollte sich jeder seinen eigenen Branch erstellen und diesen nach Abschluss der eigenen

Aufgaben auf den neuen Developer-Branch namens "targetbranch" mergen. Am Ende jedes Sprints würde dann der Developer-Branch mit dem Master-Branch gemerged werden.

IV.10 Abschließende Einschätzung des Product-Owners

Autor: xxx

XXX

IV.11 Abschließende Einschätzung des Software-Architekten

Autor: xxx

XXX

IV.12 Abschließende Einschätzung des Team-Managers

Autor: xxx

XXX

V. Sprint 2

V.1 Ziel des Sprints

Autor: Linus Herterich

Nachdem im ersten Sprint hauptsächlich die Grundstruktur sowie erste Datenstrukturen entworfen wurden, war es nun wichtig, dass sich das gesamte Team im Sprint 2 mit der Projektstruktur (besonders mit dem Framework Vue) auseinandersetzt und erste UserStories direkt am Code umsetzt. Zudem blieben einige Tickets noch vom letzten Sprint offen, welche nun auch bearbeitet werden sollten.

V.2 User-Stories des Sprint-Backlogs

Autor: Linus Herterich

- **Designumsetzung nach Adobe Preview**
  Als Benutzer der WebApplikation möchte ich ein ansehnliche und intuitive Oberflächengestaltung haben, damit ich die Applikation gerne verwende.

- **Authentifizierung eines Nutzers**
  Als Nutzer möchte ich mich in die Web Applikation einloggen können, damit nicht jeder meine erzeugten Graphen einsehen kann.

- **Logische verknüpfung zwischen Knoten erstellen**
  (wurde in Sprint 1 nicht abgeschlossen)
  Ein Nutzer muss eine Abfolge der Knoten definieren können, damit ersichtlich wird welcher Produktionsschritt auf den nächsten folgt

- **Berechnung der Eingenschaften des Gesamtgraphs**
  (wurde in Sprint 1 nicht abgeschlossen)
  Ein Nutzer der Webanwendung VarG muss die berechneten gesamt Eigenschaften jedes Zusammenhängendes Pfades ausgeben lassen können um eine Auswahl eines Pfades zu treffen.

- **Datenstruktur Ausarbeiten & Knoten zu einer vorhandenen Datenstruktur hinzufügen**
  (wurde in Sprint 1 nicht abgeschlossen)
  Als Nutzer möchte ich Knoten zu der Datenstruktur hinzufügen können um die möglichen Produktionsschritte des Werkstücks überblicken zu können

V.3 Liste der durchgeführten Meetings

Autor: Linus Herterich

- 19.12.2019: Planning Meeting
- 23.12.2019: Daily Meeting (in Discord)
- 28.12.2019: Daily Meeting (in Discord)
- 05.01.2020: Review Meeting
- 06.01.2020: Retrospektive

V.4 Ergebnisse des Planning-Meetings

Autor: Linus Herterich

Neben der Aufgabenverteilung wurde im Planning darüber gesprochen, dass die Arbeitsaufteilung im letzten Sprint nicht gut geklappt hat. Es wurde anschließend beschlossen im nächsten Sprint die User-Stories direkt an Studenten zuzuweisen, damit jeder einen Teilbereich hat, den er bearbeiten muss.
Desweiteren wurde eine Änderung im Git angekündigt. In Zukunft müsse der "Master"-Branch während eines Sprints immer gleich bleiben und Funktionalitäten werden auf einen "Developer"-Branch gemerged. Am Ende des Sprints wird dann der "Developer"-Branch auf den "Master"-Branch gemerged. wichtig ist, dass der "Master"-Branch zu jedem Zeitpunkt lauffähig ist.
Für den folgenden Sprint wurde beschlossen, die Daily Meetings online (auf einem Discord Server) abzuhalten, da viele Studenten über die Weihnachtsferien in der Heimat sind und somit ein persönliches wöchentliches treffen nicht möglich wäre.

V.5 Aufgewendete Arbeitszeit pro Person+Arbeitspaket

Autor: Linus Herterich

Arbeitspaket	Person	Start	Ende	h	Artefakt
UI: Login	Berger, Matthias	22.12.19	22.12.19	3,5	Login Funktionalität & Design
UI: Login	Buchmann, Lennart	22.12.19	22.12.19	6	Login Funktionalität & Design
UI: Grapheneditor	Gwozdz, Jonas	23.12.19	04.01.20	9	GraphHeader.vue, Toolbar.vue
Task: Einbindung in Vue-Dateistruktur	Heldt, Erik	19.12.19	19.12.19	0,25	BasicData.js
Abrufbaren Knoten in Graph einfügen	Heldt, Erik	23.12.19	26.12.19	3,5	BasicData.js, TestDatabase.js
Testdatenbank mit Speichern und Laden	Heldt, Erik	27.12.19	27.12.19	3,5	TestDatabase.js
Highlighting eines kürzesten Pfades nach Anwendung des A* Algorithmus	Henning, Tim	24.12.19	03.01.20	9	OptimizeControls.vue, index.js -> Graph Highlighting
Protokoll: Meeting 19.12.19	Herterich, Linus	19.12.19	19.12.19	1	meeting_19_12_19.pdf
UI: Login	Herterich, Linus	20.12.19	20.12.19	5	LoginForm.vue, Login.vue
UI: Home	Herterich, Linus	23.12.19	23.12.19	7	HomeMenu.vue (component), Home.vue (view), Menu.vue (view)
UI: Neuer Graph	Herterich, Linus	28.12.19	28.12.19	1,5	NewGraph.vue (view), NewGraph.vue (component)
UI: Grapheneditor	Herterich, Linus	02.01.20	04.01.20	11,75	Graph.vue (view), zahlreiche components
Graph zu Datenstruktur hinzufügen	Hohlfeld, Julius	21.12.19	23.12.19	4	BasicData.js, TestDatabase.js

Testdatenbank mit Speichern und Laden	Hohlfeld, Julius	27.12.19	03.01.20	8	BasicData.js, TestDatabase.js, index.js, JSonPersistence.js
Mergen und Anpassen	Hohlfeld, Julius	04.01.20	04.01.20	2	Bugs entfernt & Mergekonflikte behoben
UI: Datenbank-Import Fenster	Karkoutli, Alaa Aldin	31.01.20	04.01.20	12,5	Database.vue (view), DatabaseForm.vue (component)
Kanten zu Graph hinzufügen	Koch, David	23.12.20	04.01.20	10	Änderungen an index.js, CreateControls.vue (component)

V.6 Konkrete Code-Qualität im Sprint

Autor: Linus Herterich

Es wurde sich größtenteils an die Coding-Guidelines gehalten. An wichtigen Stellen sowie vor jeder Funktion wurden Kommentare geschrieben. Die Trennung zwischen Views und Components sowie die Auslagerung der Style-Dateien wurde ebenfalls eingehalten.

V.7 Konkrete Test-Überdeckung im Sprint

Autor: Linus Herterich

Ein Student wurde beauftragt bis zum Ende des Sprints ein geeignetes Test-Framework zu finden. Somit wurden während des Sprints noch keine Tests geschrieben.

V.8 Ergebnisse des Reviews

Autor: Linus Herterich

Es wurden fast alle UserStories umgesetzt. Somit war der zweite Sprint erfolgreich. Alle Studenten konnten sich in das Projekt einarbeiten und haben die Strukturierung größtenteils verstanden und eingehalten.
Das User-Interface wurde nach der Designvorlage umgesetzt und die ersten Graphen-Funktionen (Hinzufügen von Knoten und Kanten & Optimieren) funktionieren bereits.
Da noch nicht feststeht, wo die Software gehostet werden soll und wie die Datenbank-Funktionalität umgesetzt werden soll, wurde zunächst eine lokale Speicherlösung als "Datenbank"verwendet. Somit konnten die Speichern- und Laden-Funktionen erfolgreich implementiert werden.
Die Login-Funktionalität ist derzeit nur sporadisch eingerichtet und wird finalisiert, sobald feststeht, wie die Authentifizierung der Nutzer erfolgen soll (Anbindung an HTWK Login?).
Leider ist immernoch kein geeignetes Testframework gefunden worden, mit dem sich sowohl Vue.js als auch cytoscape (Graphen-Funktionalitäten) testen lassen.

V.9 Ergebnisse der Retrospektive

Autor: Linus Herterich

Das Happiness-Barometer für diesen Sprint ist sehr gut ausgefallen. Das liegt hauptsächlich an der guten Aufgabenverteilung sowie an den großen Erfolgen, die diesen Sprint erzielt wurden.
Kritisiert wurde die die Kommunikation gegen Ende des Sprints. Das finale Mergen aller Branches war zu hektisch und unsicher.
Es wurde sich darauf geeinigt in Zukunft zwei Dailies pro Woche abzuhalten und das letzte Meeting eines Sprints zum gemeinsamen Mergen zu verwenden.

V.10 Abschließende Einschätzung des Product-Owners

Autor: xxx

XXX

V.11 Abschließende Einschätzung des Software-Architekten

Autor: xxx

XXX

V.12 Abschließende Einschätzung des Team-Managers

Autor: xxx

XXX

VI. Sprint 3

VII. Sprint 4

VII.1 Ziel des Sprints

Autor: Jonas Gwozdz

Während der Semesterferien haben wir an Sprint 4 weitergearbeitet. Dieser dauerte vom 23.01.2020 bis zum 09.04.2020. Der Ablauf war dabei weitestgehend planmäßig, bis auf dass die Meetings zum Review und der Retrospektive wegen Corona ohne persönliches Treffen stattfinden mussten. In der Vorlesungsfreien Zeit besprachen wir uns gelegentlich über den aktuellen Zwischenstand. Der größte Fortschritt am Projekt wurde während der letzten beiden Wochen erzielt.

VII.2 User-Stories des Sprint-Backlogs

Autor: Jonas Gwozdz

- **Tests für bereits geschriebenen Code**
  Als Benutzer möchte ich eine Software benutzen, die getestet ist, damit keine unerwarteten Probleme auftauchen.

- **Validierung der möglichen Eingaben**
  Als Nutzer möchte ich bei versehentlicher falscher Eingabe wenn möglich gewarnt werden, damit ich nichts falsches abspeichere.

- **Bug: Validation bei gleichem Knoten-Namen**

- **Darstellung von Kanten/Attributen**
  Als Benutzer will ich alle Kanten/Knoten gleichzeitig sehen können(nicht übereinander), damit ich einen schnelleren Überblick über das gesamte Konstrukt bekomme.

- **Bug: Mehrere Edges zwischen Knoten nicht möglich**
  Wenn man mehrere Kanten zwischen zwei Knoten anlegt, sind diese nicht sichtbar. Löscht man dann einen Knoten, an dem diese ünsichtbaren"knoten hängen, so stürzt cytoscape ab.

- **Remodel von Component NewGraph**

VII.3 Liste der durchgeführten Meetings

Autor: Jonas Gwozdz

- 23.01.2020: Planning
- 05.03.2020: Weekly
- 12.03.2020: Weekly
- 06.04.2020: Review
- 09.04.2020: Retro

VII.4 Ergebnisse des Planning-Meetings

Autor: Jonas Gwozdz

Anwesend: Alex, Julius J., Julius H., Linus, Jonas, Erik, Lennart, Nils, Tim, David, Matthias, Manuel

Innerhalb dieses Meetings haben wir die Schwerpunkte des Sprints festgelegt und über den Workload über die Vorlesungsfreie Zeit diskutiert und den Zeitaufwand der User-Stories abgeschätzt.

oberste Priorität: Tests

Da wird bis zum bisherigen Zeitpunkt keine Testumgebung gefunden haben, die sich auf unseren Cytoscape-Graphen anwenden lässt, und wir dadurch viel Nachholbedarf in Sachen Testen hatten, musste dieses Ticket am dringendsten abgearbeitet werden.

Sprint über Semesterferien

Wir haben uns im Planning darauf geeinigt, den Sprint über die Semesterferien mit weniger User-Stories als üblich auszulegen, da nicht alle Teammitglieder in dieser Zeit voll verfügbar waren, Grund dafür waren vor Allem die noch andauernden Prüfungen und die Anschließenden Ferien, die evtl. schon anderweitig verplant waren. Zudem haben wir uns darauf geeinigt, regelmäßig Absprache über den Fortschritt unserer Arbeit zu halten.

Datenbanken

Die Datenbankrecherche hat ergeben, dass für unsere Zwecke mySQL oder NodeJS am optimalsten wäre. Die Definition der Datenbankschnittstelle zwischen DB und Frontend muss ebenfalls noch erledigt werden. Zudem haben wir festgestellt, dass die Bisher entworfene Datenbankoberfläche optisch nicht zum Rest der Anwendung passt, und deshalb überarbeitet werden muss.

Weitere Sprintziele:

- Optimierung der Kostendarstellung
- negative Zahleingaben abfangen
- automatische Zoomfunktion bei Knoten- oder Kantenwahl
- allgemeine Bugfixes

VII.5 Aufgewendete Arbeitszeit pro Person+Arbeitspaket

Autor: Jonas Gwozdz

Arbeitspaket	Person	Start	Ende	h	Artefakt
Tests für bereits geschriebenen Code	Heldt, Erik	04.03.20	04.03.20	2	Tests für ModifyDataControls.vue
Neue Strukturierung	Heldt, Erik	26.01.20	26.01.20	1	Umstrukturierung des Projekts
Header Buttons und Metadaten-Speicherung	Heldt, Erik	05.03.20	12.03.20	6,75	GraphHeader.vue
Aufräumen der Branches im GitLab	Heldt, Erik	29.03.20	29.03.20	1	Organisatorische Aufgabe
Entfernen veralteter Komponenten und Methoden	Heldt, Erik	31.03.20	31.03.20	2	Organisatorische Aufgabe

Tests für Graphoptimierung	Henning, Tim	04.04.20	40.40.20	12	vargraph.spec.js
Tests für bereits geschriebenen Code	Herterich, Linus	30.01.20	12.02.20	7,5	/code/cypress/integration/...
Header Buttons und Metadaten-Speicherung	Herterich, Linus	28.03.20	31.03.20	2,25	/vargraph/graph/... & GraphHeader.vue
Aufräumen der Branches im GitLab	Herterich, Linus	30.03.20	30.03.20	1	Organisatorische Aufgabe
Darstellung von Kanten/Attributen	Herterich, Linus	03.04.20	03.04.20	2	VarGraph.vue
Remodel von Component NewGraph	Herterich, Linus	30.03.20	30.03.20	3	/vargraph/graph/...
Refactoring	Herterich, Linus	29.03.20	30.03.20	9	/vargraph/graph/...
Validierung: Login	Herterich, Linus	31.03.20	30.03.20	1,5	/components/login/LoginForm
Einheitliche Alerts	Herterich, Linus	31.03.20	31.03.20	3	Dialogs.vue
Validierung CreateControls & DetailControls	Herterich, Linus	31.03.20	01.04.20	5,5	CreateControls.vue & DetailControls.vue
Bug: Mehrere Edges zwischen Knoten nicht möglich	Herterich, Linus	01.04.20	01.04.20	2	/vargraph/graph/...
Knoten dort erstellen, wo rechtsklick passiert	Herterich, Linus	01.04.20	01.04.20	1,5	/vargraph/graph/...
keybinds für Menüs	Herterich, Linus	02.04.20	02.04.20	1	
Keine Knoten aufeinander schieben	Herterich, Linus	02.04.20	02.04.20	3	/vargraph/graph/...
Einstellungsmenü erstellen	Herterich, Linus	03.40.20	05.04.20	5,5	
Tests für bereits geschriebenen Code	Hohlfeld, Julius	05.02.20	04.03.20	10	ZoomControls.spec & SaveMenu.spec & NewGraphMenu.spec & DownloadMenu.spec
Dialogfenster für Speichern, Laden und Export	Hohlfeld, Julius	24.01.20	24.01.20	2	Toolbar.vue
Validierung der möglichen Eingaben	Hohlfeld, Julius	06.04.20	06.04.20	2	divers
Refactoring	Hohlfeld, Julius	31.03.20	31.03.20	2	/vargraph/graph/...
Testing für Kanten hinzufügen	Koch, David	22.03.20	02.04.20	5	addEdges.spec

VII.6 Konkrete Code-Qualität im Sprint

Autor: Jonas Gwozdz

Die Codequatlität im allgemeinen wurde während des Sprints erheblich durch das Refactoring verbessert. Zudem wurden in nahezu allen Dateien einleitende Kommentare geschrieben, um die

zukünftige Identifizierung der gebrauchten Dateien schneller und übersichtlicher zu gestalten.

VII.7 Konkrete Test-Überdeckung im Sprint

Autor: Jonas Gwozdz

Die geschriebenen Cypress-Tests decken bereits eine Vielzahl an Funktionalitäten des Programms ab. Dazu zählen die Buttons für die Database, den Download, das Ausloggen. Zudem wurde getestet: der Speicherdialog, die Zoomeinstellungen, der Header des Graphen, das Hinzufügen von Knoten und das Erstellen eines neuen Graphen.

VII.8 Ergebnisse des Reviews

Autor: Jonas Gwozdz

Anwesend: David, Erik, Julius J., Julius H., Jonas, Linus, Manuel, Matthias, Tim

Im Rahmen des Reviews haben wir wie gewohnt die Ergebnisse des Sprint bewertet und Schwierigkeiten besprochen.

generelle Schwierigkeit: Testen
Um unsere Programm zu testen, entschieden wir uns für das Framework "Cypressëntschieden. dieses bietet End-to-End Testing an, welches allerdings nur Ausgaben des Programms auswerten kann, und deshalb sozusagen keinen Blick unter die Haube zulässt, und somit eventuell Fehler unentdeckt bleiben.

David:

- Tests für Knotenfunktionalität geschrieben
- mit Kantentests begonnen

Erik:

- Data Controls durch Header Buttons ersetzt
- Editierungsfenster entfernt
- Header Buttons getestet

Jonas:

- Testübersicht erstellt
- Möglichkeit zum Informationsaustausch über Lücken und Bugs in Tests bereitgestellt

Julius H.:

- Tests für Toolbar, Zoom-Controls, Buttons und Eingabereihenfolgen geschrieben

Julius H, Erik, Linus:

- Refactoring des Graphen, Bugfixing und Validierung von Eingaben

Linus:

- Dialogue-Popups erstellt
- Kürzelgenerierung implementiert

- Knotenüberlagerung unterbunden, Mindestabstand implementiert
- Einstellungsmenü erstellt und Implementation begonnen
- Recherche zu Datenbankfenster

VII.9 Ergebnisse der Retrospektive

Autor: Jonas Gwozdz

Anwesend: Alex, Erik, Julius J., Julius H., Jonas, Linus, Matthias, Tim

Zu Beginn des Sprints gab es keine Fortschritte zu vermelden, da vorerst die Prüfungen zu überstehen waren. In den beiden Wochen vor Sprintende wurden allerdings die wichtigsten User-Stories und sogar etwas mehr abgearbeitet.

Positiv	Negativ
-produktive Endphase -viel Motivation bei Einigen	-anfangs keine Kommunikation - wenig Motivation bei Einigen -vereinzelt Tests ohne Sinn -ausgefallene Meetings

VII.10 Abschließende Einschätzung des Product-Owners

Autor: xxx

XXX

VII.11 Abschließende Einschätzung des Software-Architekten

Autor: xxx

XXX

VII.12 Abschließende Einschätzung des Team-Managers

Autor: xxx

XXX

VIII. Dokumentation

VIII.1 Handbuch

Autor: xxx

XXX

VIII.2 Installationsanleitung

Autor: Erik Heldt

Für User:

VarG ist eine plattformunabhängige Webanwendung, das heißt man muss nichts lokal auf seinem PC installieren, um sie zu benutzen. Alles was man benötigt, ist ein moderner Web-Browser und eine Internetverbindung (Browser-Empfehlung: Google Chrome oder Firefox). Öffne den Browser und gib in der URL-Leiste `www.TODO-sampledomain.de` ein. Nun befindest du dich im Home-Menü von VarG und kannst loslegen!

Für Entwickler:

Um die Anwendung im Entwicklungszustand ausführen zu können, musst du Node.js und npm auf deinem PC installieren. Wie das geht erfährst du hier: `https://www.npmjs.com/get-npm`. Node.js ist eine JavaScript-Entwicklungsumgebung, die benötigt wird, um die Anwendung samt der genutzten Frameworks und Bibliotheken kompilieren und ausführen zu können. Node Package Manager, oder kurz npm, wird mit Node.js mitgeliefert und verwaltet alle installierten Pakete, die beim Bauen des Programms verwendet werden.

Weiterhin wird das Versionsmanagement-Tool Git benötigt. Den Download dafür gibt es hier: `https://git-scm.com/downloads` bzw. für Windows-Nutzer wird die Git BASH empfohlen: `https://gitforwindows.org/`.

Sind diese Tools nun installiert, musst du dir das VarG-Repository von GitLab auf deinen PC herunterladen bzw. "klonen". Um vollständigen Zugriff auf dieses Repository zu haben, musst du im GitLab dem Projekt zugeordnet sein. Besitzt du also die entsprechenden Rechte, navigiere im Terminal in einen Ordner auf deinem Rechner, in dem du das Projekt speichern willst, und gib dort den Befehl

```
git clone https://gitlab.imn.htwk-leipzig.de/weicker/varg.git
```

ein. Warte, bis das Herunterladen abgeschlossen ist, und öffne dann den neu erschienenen Ordner "varg" in einem Code-Editor (empfohlen wird Visual Studio Code).

Es sollten dort mehrere Ordner zu sehen sein, unter anderem der "code"-Ordner. Darin ist der gesamte Quellcode des Projekts enthalten. Du solltest dich also zur Ausführung des Programms immer in diesem Ordner aufhalten. Um nun den Code zu kompilieren und als Entwicklungsversion auszuführen, folge bitte diesem Tutorial: VarG-Installation von Linus Herterich (klickbarer Link, Anleitung nur für VSCode).

TODO: Installation der Anwendung und der Datenbank (Docker) auf dem HTWK-Server dokumentieren.

VIII.3 Software-Lizenz

Autor: Linus Herterich

Im folgenden werden die verwendeten Bibliotkeken und deren Lizenz aufgelistet:

- Vue.js - MIT License: Copyright (c) 2013-present Yuxi Evan You
- vuetify - MIT License: Copyright (c) 2016-2020 John Jeremy Leider
- cytoscape - MIT License: Copyright (c) 2016-2020, The Cytoscape Consortium.
- cytoscape-node-html-label - MIT License: Copyright (c) 2017 Kalugin Sergey
- cypress - MIT Licence: Copyright (c) 2015 Cypress.io, LLC
- jest - MIT Licence: Copyright (c) Facebook, Inc. and its affiliates.
- axios - MIT License: Copyright (c) 2014-present Matt Zabriskie
- darkmode.js - MIT License: Copyright (c) 2018 Nickolas
- file-saver.js - MIT License: Copyright (c) 2016 Eli Grey
- file-saver.js - MIT License: Copyright (c) 2016 Eli Grey

Da ausschließlich die MIT Lizenz verwendet wurde, werden wir auch die Software "VarG" unter der MIT-Lizenz veröffentlichen.

VarG-Lizenz:

IX. Projektabschluss

IX.1 Protokoll der Abnahme und Inbetriebnahme beim Kunden

Autor: xxx

XXX

IX.2 Präsentation auf der Messe

Autor: xxx

Poster, Bericht

IX.3 Abschließende Einschätzung durch Product-Owner

Autor: xxx

XXX

IX.4 Abschließende Einschätzung durch Software-Architekt

Autor: xxx

XXX

IX.5 Abschließende Einschätzung durch Team-Manager

Autor: xxx

XXX