Softwaretechnik Projekt: Crystal Disk Info

Computer Engineering

Neue Fassung für das Sommersemester 2020

Philipp Horländer, 566045
Konrad Münch, 565929

Dozent: Prof. Dr. Bauer

Berlin, 16. Juli 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Im Rahmen der Lehrveranstaltung Softwaretechnik des Studiengangs Computer Engineering an der HTW-Berlin werden Inhalte zur Softwarearchitektur, Modellierung, Abstraktion sowie Anforderungen an Software gelehrt. Des Weiteren werden verschiedene Notationen zur Darstellung und funktionalen Beschreibung vorgestellt. Ein weiterer Bestandteil des Moduls ist es, den angewandten Umgang mit den o.g. Lehrinhalten in einer praktischen Arbeit zu verbinden.

1.1 Aufgabenstellung

Es galt, sich eine Open-Source-Software auszusuchen, diese zu „Reverse-Engineeren“ und abzuändern. Dafür sollten verschiedene Modelle der Software angefertigt werden. Beispiele hierfür sind zum Beispiel Use-Case- und Domänenmodelle. Dies sollte mit einer Belegarbeit dokumentiert werden.

1.2 Beschreibung der Software

Der japanische Entwickler Noriyuki Miyazaki, programmierte Crystal Disk Info welches ein nützliches Open-Source-Programm für die Bewertung und Überwachung von internen als auch externen HDDs und SSDs ist. Mit Hilfe von S.M.A.R.T. [1] werden detaillierte Daten, wie zum Beispiel Seriennummer, Temperatur, Lese- und Schreibzyklen, Fehlerrate beim Lesen und Schreiben, Speicher und Puffergrößen, als auch die gesamten Betriebsstunden der Festplatten, aus den Festplatten-Controllern ausgelesen. Diese sind auf dem Controller-Chip gespeichert und werden in einem einstellbaren Intervall von der Software ausgelesen. Die entsprechenden Daten werden auf dem jeweiligen Host-System in Form von CSV-Dateien abgelegt.

Wie und welche Daten angezeigt werden lässt sich einstellen. Beispielsweise lässt sich die Seriennummer ausblenden, um auf Screenshots nicht aufzutauchen.
Auch lässt sich ein Diagramm über ausgewählte, gespeicherte Daten sowie Datenträgerwerte in Abhängigkeit der Zeit anzeigen.

Eine weitere nützliche Funktion für Administratoren ist es, sich bei zu hohen Temperaturen oder einer geringen Lebenszeit der Festplatten benachrichtigen zu lassen. Am lokalen Rechner wird beim Auftreten einer Warnung eine Melodie abgespielt. Des Weiteren ist es möglich, eine Warnung via E-Mail automatisch zu versenden. Wenn gewünscht lassen sich auch Farbschemata der Buttons und Anzeigen ändern.

[1] Die „Self-Monitoring, Analysis, Reporting Technology“ kurz S.M.A.R.T. ist ein Industriestandard, welcher zur Überwachung und Vorhersage eines möglichen Ausfalls des Speichermediums dient.

1.3 Visionen und Ziele des Originalprojektes

Nachdem die Recherche auf der Webseite des Entwicklers keine eindeutigen Visionen und Ziele hervorbrachte, erfolgte die Kontaktaufnahme mit Herrn Miyazaki via E-Mail. Im Folgenden wurden vom Entwickler folgende Visionen und Ziele definiert:

- **/VO1/** Crystal Disk Info soll das Risiko von Datenverlust für Menschen auf der ganzen Welt reduzieren.
- **/VO2/** Crystal Disk Info soll unter einer Open-Source-Lizenz entwickelt werden, so dass viele Leute dieses Projekt unterstützen können.
- **/VO3/** Das Programm soll Laufwerke über eine bestimmte Zeitperiode auslesen können.
- **/ZO1/** Crystal Disk Info soll S.M.A.R.T nutzen um Werte auszulesen.
- **/ZO2/** Crystal Disk Info soll über den Gesamtzustand der Festplatte informieren und bei niedriger Lebenserwartung des Laufwerkes warnen.
- **/ZO3/** Es sollen Werte und deren Verläufe visualisiert werden können.

1.4 Programm vor Änderung

Die in dem vorherigen Abschnitt genannten Funktionen lassen sich über eine gut übersichtliche Oberfläche auswählen. Auf Abbildung 1 ist das Hauptfenster mit allen Menüpunkten zu sehen, dies ist die Programmversion vor der Änderung.

2 Aufbau des Programmes

Um die gesammmte Architektur von Crystal Disk Info zusammhängend erfassen zu können, ist es relevant die Abschnitte Programmiersprache, Quellcode und Ordnerstruktur näher zu betrachten. Die Erläuterungen in den folgenden Unterabschnitten betrachten jedoch nur die wesentlichen Bereiche welche für die hier geforderte Änderung des Programms relevant sind.

2.1 Programmiersprache(n)

Crystal Disk Info wurde in der Sprache C++ für Microsoft Betriebssysteme programmiert. Zur Darstellung der grafischen Benutzeroberfläche wurde die Microsoft Foundation Classes kurz MFC[2] verwendet. Des Weiteren wurde für die Realisierung der Darstellung der Graphen die JAVA-Bibliothek **Flot**[3] verwendet.

[2]MFC ist eine von Microsoft seit 1992 entwickelte Bibliothek zur Erstellung von Windows Desktop Anwendungen.

[3]Flot ist eine reine JavaScript-Plotbibliothek für jQuery, deren Schwerpunkt auf einfacher Verwendung, attraktivem Aussehen und interaktiven Funktionen liegt. Quelle: *www.flotcharts.org*

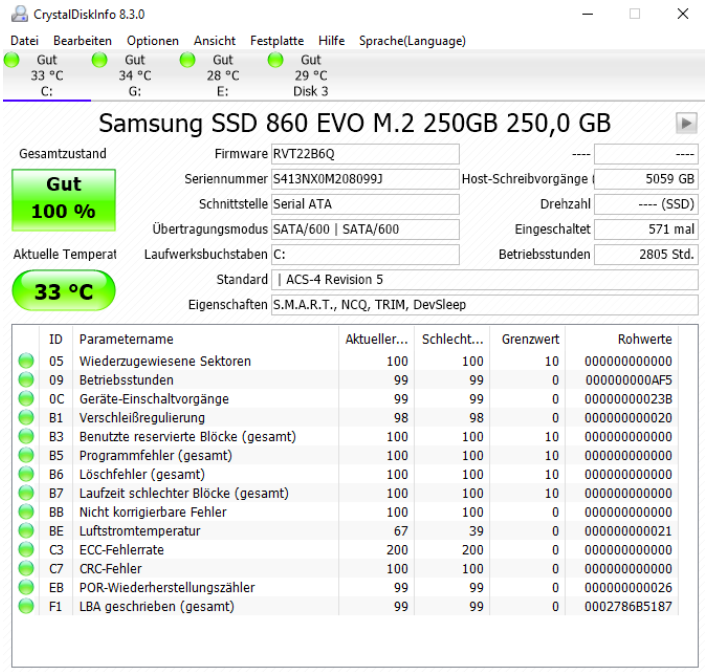


Abbildung 1: Crystal Disk Info

Zudem wurden noch HTML-Dateien genutzt, welche zur Darstellung der Graphen dienen. Durch die Diagnosefunktion von GitHub lies sich feststellen, dass C++ mit 94,7% die meist verwendete Programmiersprache in Crystal Disk Info ist.

2.2 Übersicht des Quellcodes

Zum Anfang des Projekts wurden von uns die Analyse-Tools **Sourcetrail** und **Doxygen** genutzt, um eine Übersicht vom Umfang und Beschaffenheit des Quellcodes zu bekommen. **Doxygen** erwies sich als zu mächtig und umfangreich für unsere Zwecke. Die erzeugte RTF-Datei umfasste zwar jegliche Art von Diagrammen, jedoch war die Datei mit über 32500 Wörtern nicht praktikabel einsetzbar.

Sourcetrail erwies sich hingegen als komfortabler und übersichtlicher. Wir konnten mit Hilfe dieses Tools einfacher den Aufbau und die Beschaffenheit der Klassen und Funktionen erfassen. Die Auflistung auf Abbildung 2 gibt Aufschluss über alle wesentlichen Bestandteile des Codes.

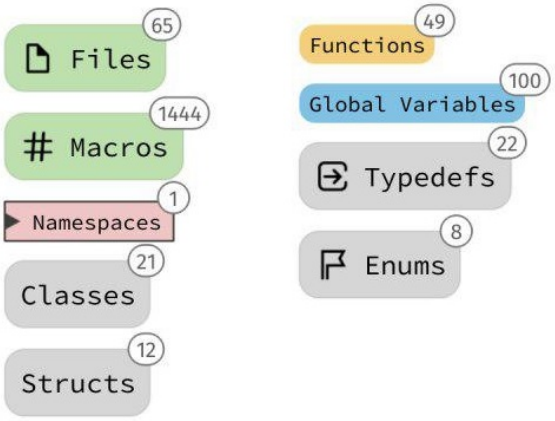


Abbildung 2: Sourcetrail Übersicht

Insgesamt umfasst der Code 34207 Zeilen in 65 Dateien. Wie unter **Abschnitt 2.1** schon erwähnt, wurde MFC für den Aufbau des Programms als Grundlage genutzt. Durch die Analyse mit **Sourcetrail** konnten wir uns einen Überblick über die wichtigste Ursprungsklasse, von der alle Elemente des GUI[4] abhängen, verschaffen. Aus der Klasse `CWinApp` werden alle erstellten Klassen wie Dialogfenster und Fensterfunktionen abgeleitet.
Aufgrund der Größe des Programms beschränkten wir uns zunächst nur auf die Klasse `CWinApp` und der des Hauptdialogs `CDiskInfoApp` (Abbildung 3). Letztere ist die von dem Autor erstellte Elternklasse aus der sich alle weiteren Dialoge des GUI ableiten.

[4]Grafische Benutzeroberfläche oder auch grafische Benutzerschnittstelle (Abk. GUI von englisch graphical user interface) bezeichnet eine Form von Benutzerschnittstelle eines Computers. Sie hat die Aufgabe, Anwendungssoftware auf einem Rechner mittels grafischer Symbole, Steuerelemente oder auch Widgets genannt, bedienbar zu machen. Quelle: *www.wikipedia.org*

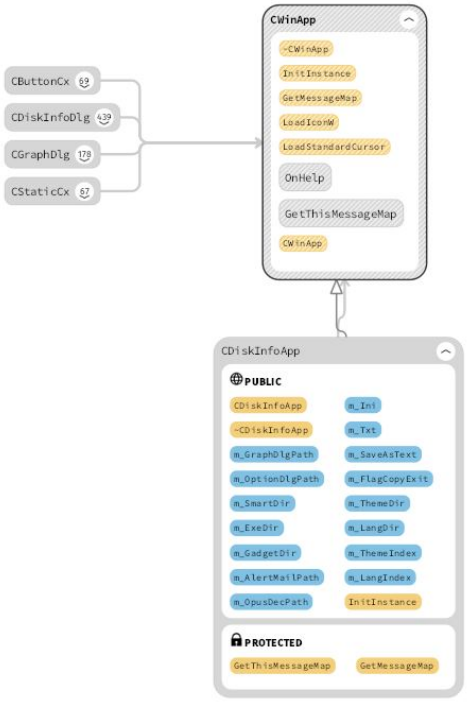


Abbildung 3: Sourcetrail Übersicht von CWinApp und CDiskInfoApp

2.3 Ordnerstruktur

Das Programm erzeugt als Root-Verzeichnis den Ordner `Marginality` . Dieser enthält die EXE-Datei und die Ordner `CdiResource` sowie `Smart`, zu sehen in Abbildung 4.
`CdiResource` enthält wichtige externe Dateien wie Sprachdateien, Themen, die JavaScript-Dateien für den Graphen sowie das Programm `AlertMail` welches genutzt wird, um Warnungen via E-Mail zu versenden.

Der Ordner `Smart` enthält die durch Crystal Disk Info ermittelten Werte der verbauten Festplatten im Host-System. Diese Daten werden vom GUI und der JavaScript-Bibliothek **Flot** genutzt, um die Werte im Frontend des Programmes darzustellen.

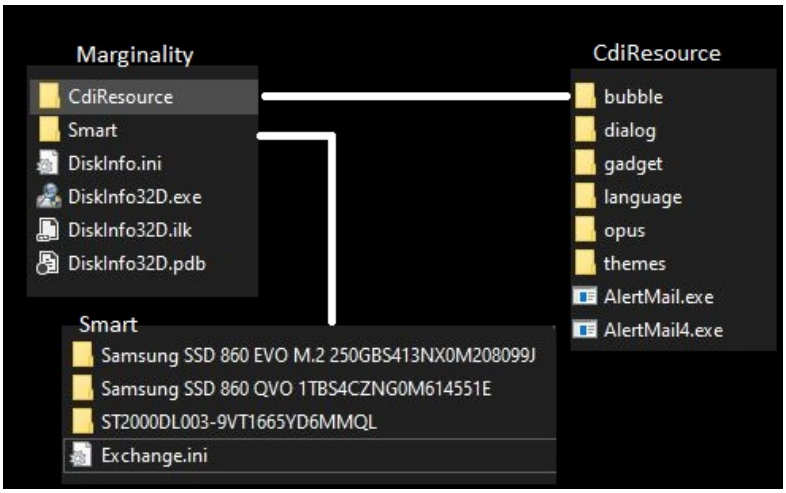


Abbildung 4: Ordner Struktur im Programm (nicht Source-Code)

Ohne diesen Aufbau ist das Programm nicht lauffähig. Das GIT-Repository von Crystal Disk Info enthält nicht den vollständigen Ordneraufbau. Es fehlt der Ordner `CdiResource`, dies führt nach einem Compiler-Durchlauf zu Laufzeitfehlern. Bei der Installation mittels Setup-Assistent werden alle nötigen Verzeichnisse erstellt.

3 Modellierung

Zur formalen Darstellung des Funktionsumfangs von Crystal Disk Info, werden unter diesem Abschnitt zwei Notationen verwendet. Das sogenannte Use-Case-Modell, welches auf Abbildung 5 dargestellt ist, dient als Abstraktion, um aufzuzeigen welche Möglichkeiten dem Nutzer zur Verfügung stehen. Es informiert zudem über weitere Relationen wie z.B. zu externen Aktoren.

Des Weiteren wird mit dem Domänenmodell eine Übersicht der Klassen und Funktionen des Programms gegeben. Unter dem **Abschnitt 3.2** auf Abbildung 6 befinden sich die relevanten Informationen.

3.1 Use-Case-Modell

Das hier gezeigte Use-Case-Modell gibt Aufschluss über alle Anwendungsfälle. Die einzelnen Abschnitte informieren über die Abläufe und alternativen Ausgangsmöglichkeiten für jeden Fall.

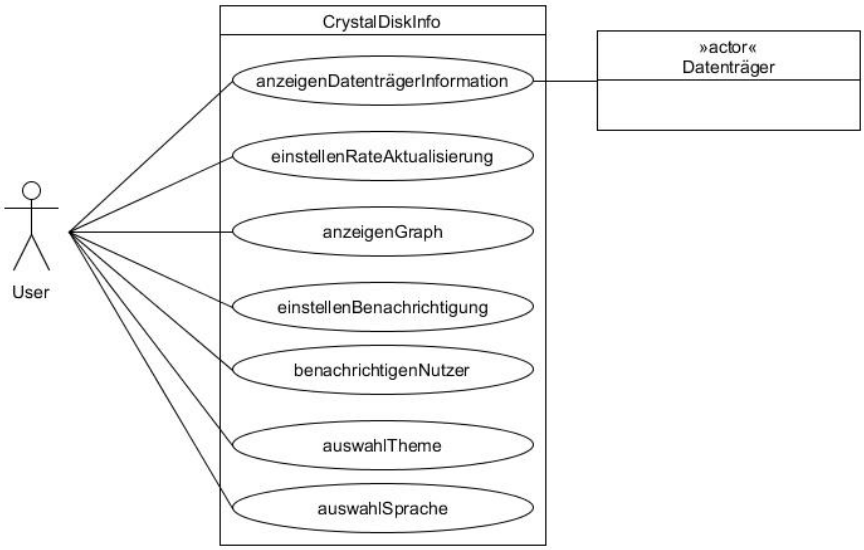


Abbildung 5: Use-Case-Modell

3.1.1 anzeigenDatenträgerInformation

1. Das Programm ermittelt die verbauten Festplatten.
2. Das Programm liest S.M.A.R.T.-Daten jedes Festplatten-Controllers aus (**/ZO1/**).
3. Das Programm zeigt die ermittelten Werte im GUI an (**/ZO3/**).

3.1.2 einstellenRateAktualisierung

1. Der Nutzer wählt ein Zeitintervall.
2. Das Programm liest und speichert die Werte im gewählten Intervall.

3.1.3 anzeigenGraph

1. Der Nutzer wählt die Funktion Graph anzeigen (**/ZO3/**).
2. Das Programm öffnet ein neues Fenster.
3. Der Nutzer wählt eine Datenreihe zur Anzeige aus.
4. Das Programm zeigt einen Graph mit gewählten Datenreihen an (**/ZO3/**).

2a. Der Nutzer wählt eine andere Datenreihe aus.
2a.1. Das Programm zeigt die andere Datenreihe an (**/ZO3/**).

2b. Der Nutzer wählt Festplatten an/ab.
2b.1. Das Programm zeigt die gewählten Festplatten-Datenreihen an (**/ZO3/**).

3.1.4 einstellenBenachrichtigung

1. Der Nutzer wählt Einstellen einer Benachrichtigung aus.
2. Das Programm wartet Eingabe der benötigten Daten des Benutzers.
3. Der Nutzer gibt nötigen Daten ein und bestätigt.

3.1.5 benachrichtigenNutzer

1. Das Programm stellt einen kritischen Wert von Temperatur oder Lebensdauer einer Festplatte fest (**/ZO2/**).
2. Das Programm erstellt eine E-Mail und versendet diese an die eingestellte Adresse (**/ZO2/**).

2a Der Nutzer schließt das Fenster.
2a.1 Das Use-Case endet erfolglos.

3.1.6 auswahlTheme

1. Der Nutzer wählt Theme ändern aus.
2. Das Programm zeigt dem Nutzer eine Auswahl an.
3. Der Nutzer wählt zwischen Verschiedenen Farbdarstellungen.
4. Das Programm lädt die ausgewählte Darstellung.

(2-3)a Der Nutzer bricht die Auswahl ab.
(2-3)a.1 Das Use-Case endet erfolglos.

4‖a. Das Programm stellt fest, dass benötigte Dateien fehlen:
4‖a.1. Das Programm zeigt eine Fehlermeldung an.
4‖a.2. Das Use-Case endet erfolglos an.

3.1.7 auswahlSprache

1. Der Nutzer wählt Sprache ändern aus.
2. Das Programm zeigt dem Nutzer eine Auswahl an.
3. Der Nutzer wählt zwischen verschiedenen Sprachen.
4. Das Programm lädt die ausgewählte Sprache.

(2-3)a Der Nutzer bricht die Auswahl ab.
(2-3)a.1 Das Use-Case endet erfolglos.

4‖a. Das Programm stellt fest das benötigte Dateien fehlen:
4‖a.1. Das Programm zeigt eine Fehlermeldung an.
4‖a.2. Das Use-Case endet erfolglos an.

3.2 Domänenmodell

Das in Abbildung 6 gezeigte Domänenmodell stellt die Funktion vor der Änderung des Programmes dar. Zur Übersicht wurde nur der Ausschnitt des gesamten Programmes betrachtet, der für die spätere Änderung relevant ist.

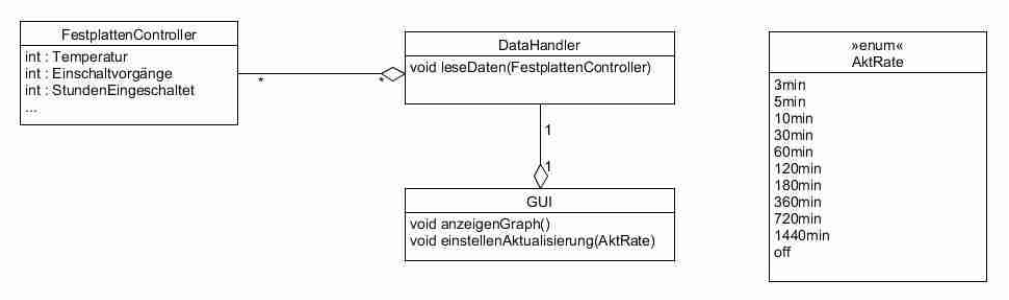


Abbildung 6: Domänenmodell: Funktionsweise

4 Geforderte Änderungen

Dieser Abschnitt informiert über die Änderungen, welche Bestandteil der praktischen Arbeit mit der Open-Source-Software waren. Zuerst werden die Visionen und Ziele unserer Änderungen kurz definiert, des Weiteren werden diese im Detail erläutert.

4.1 Visionen und Ziele der Änderung

Anfänglich gab es viele Visionen, welche in der näheren Auswahl waren, um als Änderung umgesetzt zu werden. Letztendlich entschieden wir uns in Absprache mit dem Dozenten für folgende zwei Ziele:

- **/ZÄ1/** Crystal Disk Info soll über einen neuen Diagrammtyp verfügen, welcher drei Parameter auf einem zweidimensionalen Graphen abbilden kann.
- **/ZÄ2/** Crystal Disk Info soll sekündlich die Aktualisierung der ausgelesenen Werte ermöglichen.

Die Änderung von Crystal Disk Info sollte die Darstellung eines neuen Diagrammtyps (**/ZÄ1/**) ermöglichen. Das sog. „Bubble Diagramm“ sollte in die Auswahlfunktion der Diagramme implementiert werden.

Ein „Bubble Diagramm“ soll die Korrelation von zwei Datenreihen in Form von Punkten auf einem Koordinatensystem darstellen. Diese werden durch eine dritte Abhängigkeit ergänzt, welche durch die Größe der Punkte (Radius) dargestellt wird.

Eine Umsetzungsidee war die Darstellung von Temperatur auf der X-Achse zu geschriebenen Daten auf der Y-Achse, welche nach einer Zeitperiode ihre Größe verändern. Zudem sollten noch andere Daten mit einander verglichen werden.

Nach Absprache mit dem Dozenten Herrn Prof. Dr. Baar wurde die Art der Implementierung vereinfacht. Grund hierfür war die Komplexität des Aufbaus der Darstellungsfunktion im Programm.
Eine weitere Änderung sollte darin bestehen, die Aktualisierungszeit (**/ZÄ2/**) der Datenpunkte auf eine Sekunde hinzuzufügen.

5 Modellierung nach den Änderungen

Die folgenden Unterabschnitte dieses Kapitels, befassen sich mit dem modifizierten Use-Case-Modell, der neu eingeführten Klasse und dem aktualisierten Domänenmodell.

5.1 Use-Case-Modell

Änderungen gab es an zwei Use-Cases, diese wurden auf Grundlage der Anforderungen von **Abschnitt 4.1** umgesetzt.

1) **anzeigenGraph**

   Das Programm generiert eine HTML-Datei und öffnet diese im Browser (**/ZÄ1/**).

2) **einstellenRateAktualisierung**

   Das Programm wurde um die Funktion, die Aktualisierung der Laufwerkswerte sekündlich durchzuführen, erweitert (**/ZÄ2/**).

5.2 Neue Klasse

Um das „Bubble Diagramm“ zu erzeugen wurde eine neue Klasse `BubblePlotHandler` (Abbildung 7) angelegt. Die Klasse besitzt eine öffentliche Methode `makeHTML()`, eine private Methode `getDataFromFile(path)` und keine Attribute.

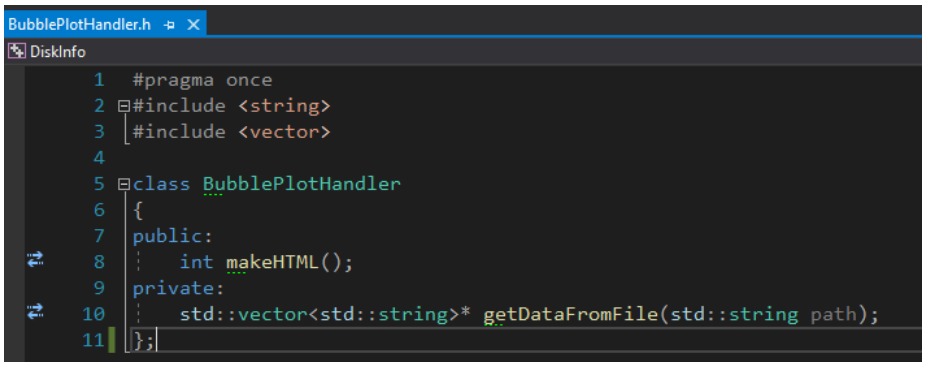


Abbildung 7: BubblePlotHandler.h

Wird `makeHTML()` aufgerufen, werden Daten aus CSV-Dateien ausgelesen (Abbildung 8). Wir konnten nicht herausfinden, wie der Pfad zu den Festplattendaten auszulesen ist, deshalb liest die Klasse Beispieldaten aus, die im selben Ordner liegen (`Smart/testData`).[5]

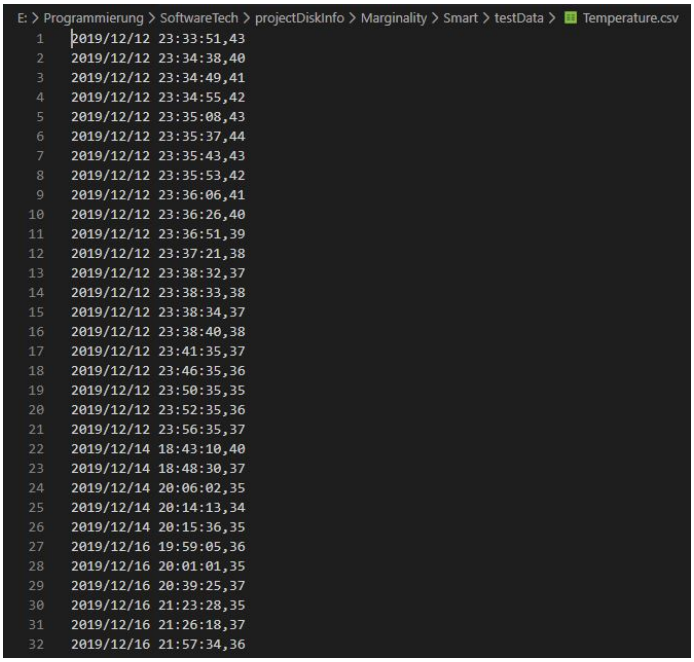


Abbildung 8: Beispiel für Datenreihe in CSV-Datei

[5]Die Daten liegen als ZIP-Archiv (`testData.zip`) vor und müssen manuell eingefügt werden, da Crystal Disk Info den Smart Ordner frei räumt, wenn es auf einem anderen PC gestartet wird.

5.3 Domänenmodell

Das hier auf Abbildung 9 dargestellte Modell, zeigt die neu eingeführten Funktion `anzeigenBubbleDiagramm` und die neue Aktualisierungsrate als `enum` von einer Sekunde.

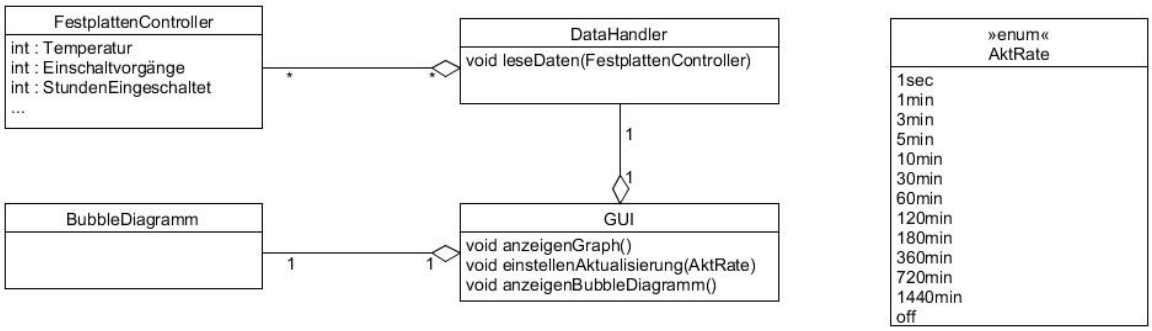


Abbildung 9: Domänenmodell: Funktionsweise nach der Änderung

6 Beschreibung der Implementierung

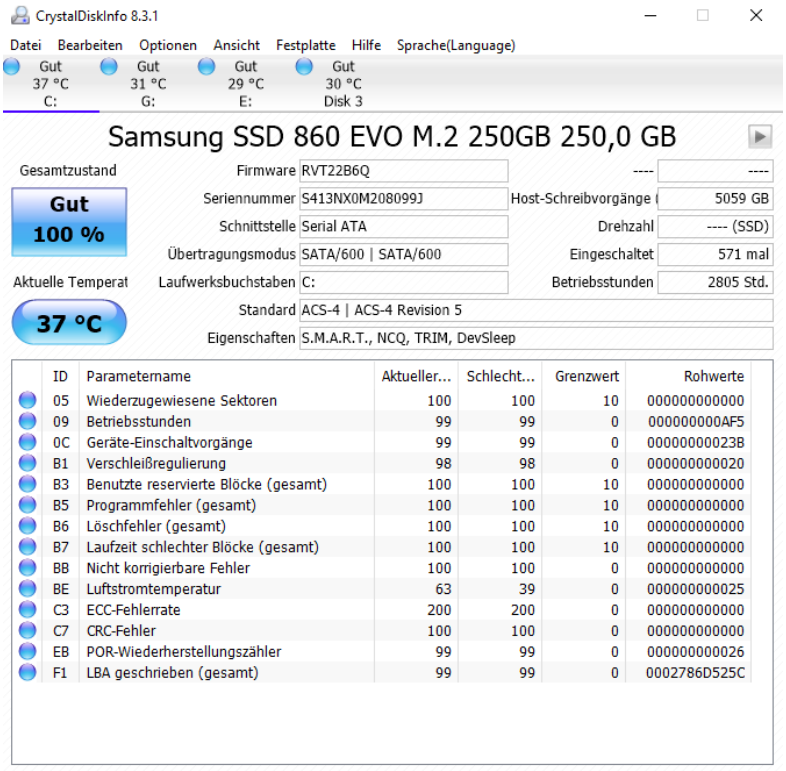
Um das „Bubble Diagramm“ darzustellen wurde, wie im Originalprogramm, eine JavaScript Bibliothek genutzt. Es wurde sich allerdings für eine andere, modernere Bibliothek entschieden, da die in Crystal Disk Info verwendete schon sehr alt war und die Anforderung, ein „Bubble Diagramm“ darzustellen, nicht erfüllen konnte. Deshalb wurde die JavaScript-Bibliothek `ChartsBundle.js` verwendet, welche als Open-Source-Datei auf GitHub zur Verfügung steht.

Mithilfe dieser Bibliothek wurde dann eine HTML-Datei mit JavaScript erstellt. Um die Verarbeitung in C++ zu vereinfachen, wurde das Skript in zwei Teile geteilt und in Textdateien abgelegt. Diese Dateien wurden dann in C++ mit den ausgelesen Daten zu einer fertigen HTML/JS-Datei verknüpft und gespeichert. Die erstellte HTML-Datei wird nun über einen Aufruf im Browser geöffnet.

7 Programm nach Änderung

Die implementierten Änderungen sind auf den ersten Blick auf Abbildung 10 nicht sofort ersichtlich. Das Hinzufügen der Aktualisierungsrate der Festplattenwerte mit dem Wert von einer Sekunde ist unter dem Menüpunkt *Ansicht* auswählbar. Die neue Diagrammfunktion verdrängte nicht die schon vorher verfügbaren Darstellungen. Das „Bubble Diagramm“ existiert nun parallel zu der zuvor genannten Diagrammfunktion.

Um diese Darstellung aufrufen zu können, ist ebenfalls der Menüpunkt *Ansicht* anzuwählen. Der Nutzer hat nun unter einem Unterpunkt namens *Charts*, die Möglichkeit diese neue Funktion auszuwählen und sich darstellen zu lassen. Auf Abbildung 11 ist diese neue Option sichtbar.



ID	Parametername	Aktueller...	Schlecht...	Grenzwert	Rohwerte
05	Wiederzugewiesene Sektoren	100	100	10	000000000000
09	Betriebsstunden	99	99	0	000000000AF5
0C	Geräte-Einschaltvorgänge	99	99	0	00000000023B
B1	Verschleißregulierung	98	98	0	000000000020
B3	Benutzte reservierte Blöcke (gesamt)	100	100	10	000000000000
B5	Programmfehler (gesamt)	100	100	10	000000000000
B6	Löschfehler (gesamt)	100	100	10	000000000000
B7	Laufzeit schlechter Blöcke (gesamt)	100	100	10	000000000000
BB	Nicht korrigierbare Fehler	100	100	0	000000000000
BE	Luftstromtemperatur	63	39	0	000000000025
C3	ECC-Fehlerrate	200	200	0	000000000000
C7	CRC-Fehler	100	100	0	000000000000
EB	POR-Wiederherstellungszähler	99	99	0	000000000026
F1	LBA geschrieben (gesamt)	99	99	0	0002786D525C

Abbildung 10: Screenshot Crystal Disk Info nach Änderung

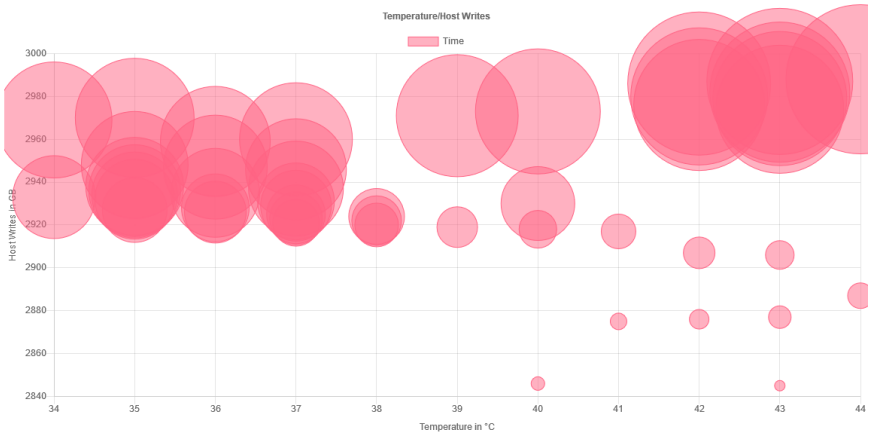


Abbildung 11: Neuer Diagrammtyp Bubble Diagramm

8 Fazit

Schwierigkeiten traten im gesamten Verlauf zur Genüge auf. Zuerst war recht schwer zu lokalisieren, wie Crystal Disk Info aus dem Source Code funktionsfähig zu bekommen ist. Nötig waren dafür Dateien von der fertigen Installation die dem Source Code nicht beilagen und auch nicht beim Compilieren erstellt wurden.

Als nächstes war es sehr schwer den Code zu verstehen, da es sich bei dem Entwickler von Crystal Disk Info offensichtlich um einen sehr erfahrenen Programmierer handelt. Weiterhin ist Crystal Disk Info sehr spärlich kommentiert in den Hauptklassen. Viele Kommentare waren auf Japanisch, was die Bearbeitung verzögerte, da wir diese erst übersetzen mussten.

Die geforderte Änderung umzusetzen erwies sich als schwerer als erwartet. Wir versuchten erst zu verstehen, wie Herr Miyazaki den Graph erstellte und stießen dabei auf eine JavaScript Bibliothek. Allerdings war auch diese Bibliothek nicht zielführend. Wir suchten uns dann einige Bibliotheken für C++ heraus, konnten aber keine verwenden, da etwaige andere Abhängigkeiten zu weiteren Bibliotheken bestanden, wie zum Beispiel zu QT (einer konkurrierenden GUI-Bibliothek zu MFC) oder zu Python.

Letztendlich verwendeten wir eine kleine JavaScript-Bibliothek. Denn wir stellten fest, dass MFC keine Möglichkeit hat JavaScript in HTML zu verarbeiten. Also bauten wir intern eine HTML-Datei, welche mit Hilfe einer JavaScript-Datei die Darstellung ermöglichte und riefen diese im Standardbrowser auf.

Durch diese vielen Probleme lernten wir, dass viele Details beim Softwaredesign zu beachten sind. Des Weiteren war eine genau Planung erforderlich. Schlussendlich wurde uns bewusst, dass sich initial einfach erscheinende Änderungen im Endeffekt aufwendiger als gedacht herausstellen können.