Abschlussbericht

9. März 2020

Hochschule Geisenheim | Institut für Gemüsebau	314-06.01-2816HS003

Optimierte Anwendbarkeit und Zugänglichkeit eines Entscheidungshilfesystems zur effizienten Bewässerungssteuerung von Freilandgemüse auf Grundlage der Geisenheimer Steuerung
GS Entscheidungs-**H**ilfe **E**infach **N**utzbar - **GSEHEN**

Laufzeit: 01.04.2016 – 31.12.2019

Berichtszeitraum: 01.04.2016 – 31.12.2019

Zusammenarbeit mit:

Dr. Sebastian Weinheimer, Versuchsbetrieb Queckbrunnerhof,
Dienstleistungszentrum Ländlicher Raum Rheinpfalz

CGS mbH - Consulting Gesellschaft für Systementwicklung
Lange Straße 1, 38100 Braunschweig

1 Ziele und Aufgabenstellungen des Vorhabens

Dem schonenden Einsatz der Ressource Wasser und der Vermeidung von Stoffeinträgen ins Grundwasser wird zunehmende gesellschaftliche Bedeutung zugemessen. Dieser gesellschaftlichen Erwartung kann die gemüsebauliche Produktion mit einer bedarfsgerechten Wasserversorgung Rechnung tragen. Als wirkungsvolles Instrument steht dafür die Geisenheimer Steuerung (GS) zur Verfügung. Sie ist ein Verfahren der Klimatischen Wasserbilanz (KWB), die den Wasserverbrauch einer Gemüsekultur berechnet, diesen mit Niederschlägen bilanziert und Bewässerungsempfehlungen ausspricht. Der Wasserverbrauch ergibt sich dabei aus einer meteorologischen Referenzverdunstung, die mit kulturspezifischen Korrekturfaktoren, den sog. Kc-Werten, multipliziert wird.

Das Ziel des Vorhabens ist, die Verfügbarkeit und Praktikabilität der GS für den Anwender zu verbessern. Gemüseanbauern erhalten dadurch ein Angebot, um die Effizienz des Einsatzes der Ressource Wasser zu erhöhen und das Erfüllen gesetzlicher Auflagen zum Wasserschutz zu erleichtern. Dies stärkt den gärtnerischen Berufsstand und erhöht dessen Wettbewerbsfähigkeit.

Mit Hilfe des modular aufgebauten Entscheidungshilfesystems GSEHEN sollen der Beratung und den Gemüseerzeugern die Nutzung der GS erleichtert werden. Anbietern von Beratungssystemen soll die Möglichkeit eröffnet werden, die GS in ihre Systeme zu integrieren, z. B. in elektronische Schlagkarteien oder in die Steuerung von Beregnungsmaschinen.

Für die Implementierbarkeit von GS in Farm Management Systeme fehlten bisher:

- eine vollständige und öffentlich zugängliche Dokumentation des Systems
- ein Standard-IT-Protokoll und entsprechende Schnittstellendefinitionen für den Zugang zu einem Datenserver
- eine verbesserte Datenaufbereitung bzw. -generierung über den kulturspezifischen Wasserbedarf und über digitale Beregnungsempfehlungen

Die Implementierbarkeit muss dabei produkt- und unternehmensneutral möglich sein. Mit den Systemkomponenten soll es Anwendern, Beratern und Serviceanbietern zukünftig möglich sein, eine GS-basierte Beregnungsempfehlung auch ohne Unterstützung durch die Hochschule Geisenheim (HGU), ohne regelmäßige Beobachtung des Pflanzenentwicklungszustandes und ohne eigene Tabellenkalkulation automatisch zu generieren.

Dazu gilt es, den aktuellen Wissenstand über die Eingangsparameter der GS zusammenzutragen und zu überprüfen. Dies umfasst zum einen pflanzenentwicklungsabhängige Kc-Werte, die das Verhältnis der tatsächlichen Verdunstung zu einer Referenzverdunstung darstellen. Zum anderen handelt es sich um die Dauer der jeweiligen Kc-Entwicklungsstadien, die entwicklungsspezifischen Durchwurzelungstiefen und die

Wasserspeicherfähigkeit verschiedener Bodenarten. Vorgegebene, von den Anwendern korrigierbare, Entwicklungsdauern werden die Anwendung der GS erheblich erleichtern und das Risiko minderen, eine notwendige Kc-Wert-Anpassung während der Entwicklung der Gemüsekultur zu versäumen. Mit den Boden- und Wurzeltiefen-Parametern lässt sich die Höhe der Einzelwassergaben schlagspezifisch optimieren, um beregnungsbedingtes Sickerwasser zu vermeiden. Alle Parameter der GS werden in eine für automatisierte Beratungssysteme geeignete Form gebracht und in einer frei zugänglichen Online-Datenbank publiziert. Die HGU kann somit sicherstellen, dass Nutzer stets den Stand des Wissens erhalten und aktuelle Parameter verwenden. Wissenschaftlich begründete Veränderungen der Parameter stehen ihnen somit ohne zeitliche Verzögerung zur Verfügung. Darüber hinaus wird das Entscheidungshilfesystem in modularer Form durch die HGU und ein Softwareunternehmen entwickelt und anschließend als FOSS (free and open source software) bereitgestellt.

Das Entscheidungshilfesystem umfasst Module für die Berechnung der Klimatische Wasserbilanz nach GS, die Aufbereitung von benötigten Wetterdaten, eine Datenbank und eine Benutzeroberfläche. Diese ermöglicht es, schlagspezifische Bewässerungen von Gemüsekulturen mit Hilfe der genannten Module zu berechnen und zu planen. Durch die Modularisierung wird eine flexible Nutzung der GS durch eine Vielfalt unterschiedlicher Nutzer ermöglicht. Anwender aus der Gemüsebaupraxis oder der Beratung können mit Hilfe dieses Systems selbständig Bewässerungsvorgänge planen. Dienstleister, wie etwa Anbieter von Schlagkarteien, können die GS durch den Zugriff auf die frei zugänglichen Module des Entscheidungshilfesystems GSEHEN in das eigene Beratungssystem integrieren, ohne dabei die Funktionalität oder Datengrundlage der GS selbst entwickeln zu müssen. Dies garantiert eine fach- und sachgerechte Berechnung der GS durch alle zukünftigen Anwender. Für die Umsetzung und Anwendung der GS wird eine Dokumentation erstellt. Diese wird alle systemrelevanten Algorithmen und klar definierten Umsetzungsstandards in Form von Programmablaufplänen beinhalten. Für die ausgewählte Kultur Zwiebel, deren Parameter sich als inkonsistent erwiesen haben, sollen methodische Fragen mit Hilfe direkter Verfahren der Evapotranspirations-Messung im Feld (Eddy-Kovarianz-Methode) beantwortet werden. Mit dieser Methode verbindet sich die Erwartung, den Wasserverbrauch von Gemüsekulturen unter Feldbedingungen exakter ermitteln zu können, als dies mit den bisherigen, in Lysimetern gewonnenen Schätzwerten gelang. Mit Hilfe dieser Ergebnisse können erstmalig Einschätzungen zum tatsächlichen Wasserverbrauch direkt unter Feldbedingungen gemacht werden. Damit wird die Qualität der Beregnungsempfehlungen der GS deutlich zu verbessern sein. Die Entwicklung von GSEHEN kann dadurch helfen, die Ressourceneffizienz zu steigern und somit den ökonomischen sowie ökologischen Ansprüchen an die Gemüseproduktion vor dem Hintergrund gesetzlicher Rahmenbedingungen wie der EU-WRRL oder Nitratrichtlinie besser gerecht werden.

1.1 Planung und Ablauf des Vorhabens

Eine bessere Praxisimplementierung der GS soll am Institut für Gemüsebau der Hochschule Geisenheim in Kooperation mit dem Versuchsbetrieb Queckbrunnerhof des Dienstleistungszentrums Ländlicher Raum Rheinpfalz (DLR-RP) und mit Hilfe eines Unterauftragnehmers aus der IT-Branche erarbeitet werden.

Dafür erfolgt eine Sichtung, Zusammenfassung und Aktualisierung bestehender Parameter der GS, das Verfassen einer Dokumentation mit Anwendungsrichtlinien, sowie die Entwicklung eines Beregnungsberatungssystems, das auf einer Online-Datenbank basiert und aus Modulen besteht. Unter Verwendung langjähriger empirischer Daten von GS, erhoben an der HGU und am DLR-RP, auf Grundlage von Erfahrungen aus dem Modellvorhaben "Demonstrationsbetriebe zur Effizienzsteigerung der Bewässerungstechnik und des Bewässerungsmanagements im Freilandgemüsebau" und dem Innovationsprojekt "GS-Mobil" wird mithilfe eines IT-Unternehmens eine server- und schnittstellen-basierte Kalkulationsplattform zur vereinfachten Anwendbarkeit der GS kreiert.

Im ersten Projektabschnitt wird von der HGU definiert, welche Anforderungen aus dem Dienstleistungsbereich Schlagkartei, Beregnungsberatung oder Beregnungsmaschinen-steuerung gestellt werden. Eine Ausschreibung, basierend auf diesem Kriterienkatalog, wird dann zur Auswahl eines IT-Unterauftragnehmers führen. Das IT-Unternehmen erstellt in Zusammenarbeit mit der HGU Standardprotokolle und definiert Schnittstellen für die Datenbankanbindung und die Module. Parallel dazu überarbeitet die HGU die Parameter der GS und evaluiert kritische Parameter. Die so aktualisierte Parametersammlung wird in einem von der HGU eingerichtetem Server in einer Datenbank gespeichert. Weiterhin werden die für das Entscheidungshilfesystem zusätzlich benötigten Module (Klimatische Wasserbilanz, Wetterdaten, Benutzeroberfläche) entwickelt. Zuletzt werden alle Systemkomponenten miteinander verknüpft und einem Funktionstest unterzogen. Resultierend soll eine freizugängliche, auf den aktuellsten Wissenstand gestützte, und auf zeitgemäßen Kommunikationsstandards basierende Dokumentation und eine Daten- und Methoden-sammlung in Form von Modulen entstehen. Dies wird die Umsetzung der GS in computergestützte Beratungssysteme erleichtern und eine einfache Anwendung der GS durch Beratung und Gemüseanbauer ermöglichen. Die HGU erstellt als Wissenslieferant die Dokumentation und Anleitung der GS sowie die Parameter kc-Werte, die Entwicklungs-dauern für verschiedene Gemüsearten, die Bodenkennwerte und die Durchwurzelungstiefen. Die HGU richtet eine Datenbank ein, die diese Parameter zur Verfügung stellt. Darüber hinaus entwickelt die HGU ein Modul zur Berechnung der GS. Der Unterauftragnehmer entwickelt die Module zur Bereitstellung der Wetterdaten sowie der graphischen Benutzeroberfläche (GUI), die eine schlagspezifische Bewässerungsplanung ermöglicht.

Das entstehende System gibt Bewässerungsempfehlungen, basierend auf den Parametern von GS und Wetterdaten sowie den anwenderspezifischen Daten zu Boden, Kultur und bereits ausgebrachten Bewässerungsmengen. Die Wetterdaten sollen dabei durch das

Wetterdaten-Modul automatisiert erhoben werden, entweder über einen externen Wetterdatenlieferanten oder über eine bei den Anwendern installierte Wetterstation. Der Anwender nimmt aufgrund der Empfehlungen Bewässerungen vor und bucht verabreichte Bewässerungsmengen in das System ein.

1.2 Wissenschaftlicher und technischer Stand, an dem angeknüpft wurde

Die Steuerung der Bewässerung erfolgt in Praxisbetrieben nach wie vor meist intuitiv, nach gärtnerischer Erfahrung oder nach Spatenprobe (Zinkernagel et al., 2012). Wenn objektive Kriterien herangezogen werden, dann erfolgt die Bewässerungssteuerung im Freilandgemüsebau weltweit hauptsächlich auf Grundlage von Bodenfeuchte-Messungen oder der Berechnung des Bodenwasserhaushaltes mit Hilfe der KWB (Pardossi and Incrocci, 2011, Cahn und Johnson, 2017). Ob Bodenfeuchtesensoren oder Kalkulationsprogramme zur KWB zum Einsatz kommen, entscheidet i.d.R. die Kulturart und die verwendete Bewässerungstechnik. Bodenfeuchte-Sensoren sind bei Tropfbewässerung zu empfehlen, die in Mitteleuropa nur bei wenigen Kulturen mit weiten Reihenabständen Bedeutung hat, z. B. bei Spargel.

Die KWB eignet sich besonders für die verbreiteten flächigen Beregnungsverfahren, wie z. B. für Rohrberegnungsanlagen oder Trommelberegnungsmaschinen. Die KWB kalkuliert den kulturspezifischen Wasserbedarf auf Grundlage einer Referenzverdunstung, des Niederschlags und sog. Kc-Werte: Diese kulturspezifischen Korrekturfaktoren berücksichtigen den variierenden Wasserverbrauch einer Kultur im Laufe ihrer phänologischen Entwicklung. Sie existieren für viele Gemüsekulturen im Rahmen von Bewässerungssteuerungssystemen wie FAO-56 und GS (Allen et al., 1998; Bryla et al., 2010; Hartmann et al., 2000; Paschold et al., 2002; Paschold et al., 2011; Zinkernagel et al., 2019). Die GS verfügt über Kc-Werte für eine Vielzahl von Gemüsekulturen für mitteleuropäische Anbaubedingungen. Zur Anwendung der KWB bedarf es einer regelmäßigen Einschätzung der Pflanzenentwicklung durch den Nutzer, um den Kc-Wert entsprechend des Entwicklungsstadiums des Pflanzenbestandes auswählen zu können.

Die bisherige Forschung (GS-Mobil: BLE, FKZ: 2815500111) fokussiert sich auf die Prognostizierung des Pflanzenwachstums auf Grundlage von Temperatursummen, um Kc-Werte zu modellieren, damit regelmäßige Beobachtungen der Pflanzenentwicklung zum Anpassen von Kc-Werten obsolet werden und das Verfahren deutlich anwenderfreundlicher wird (Olberz und Zinkernagel, 2014).

Entscheidungshilfesysteme auf Grundlage der KWB werden i.d.R. auf nationaler Ebene weiterentwickelt. Die Basis hierfür stellt häufig die international generierte und meist-evaluierte FAO-56 Richtlinie dar (Allen et al., 1998). Die FAO entwickelte ein englischsprachiges Computerprogramm, CROPWAT, für die Bewässerungsplanung und -steuerung (Smith, 1992). Eine Modifikation von CROPWAT wurde für Spanien entwickelt (Moneo and Iglesias, 2007). Beide Systeme sind allerdings sehr komplex und hauptsächlich

im wissenschaftlichen Bereich anwendbar. Praktikable Systeme existieren zum Beispiel für Griechenland mit IRMA-SYS (Tsirogiannis et al., 2018), für Spanien mit Irrigation-Advisor (Mirás-Avalos et al., 2019) oder in Italien mit GesCoN (Elia and Conversa, 2015).

Für die KWB wurden im deutschsprachigen Raum u. a. folgende Systeme entwickelt: GS (HGU), Zephyr (Ingenieurbüro Dr. R. Michel), Irrigama (Irrigama GbR). Mit agrowetter Beregnung (Deutscher Wetterdienst, DWD) steht dem Praktiker eine Entscheidungshilfe für die Bewässerung auf Grundlage der GS zur Verfügung.

Die Akzeptanz vorhandener Angebote im Praxisbetrieb leidet jedoch an der Notwendigkeit einer täglichen Eingabe und Betreuung der Programme. Das derzeitige System von GS erfordert vom Anwender regelmäßig, das schlagspezifische Pflanzenentwicklungsstadium einzuschätzen und eine täglich wiederkehrende Kalkulation des Beregnungsbedarfs vorzunehmen. Für eine breite Anwendung einer ressourceneffizienten Bewässerungs-steuerung im Gemüsebau stellt die GS jedoch nach wie vor die wichtigste Grundlage für weiterführende Entwicklungen dar.

2 Material und Methoden

Das Projekt GSEHEN verfolgt zwei Teilziele, zum einen die Softwareentwicklung des Programmes GSEHEN und zum anderen in die Überprüfung und Dokumentation der GS sowie deren Kc-Werte.

2.1 Softwareentwicklung

Die Softwareentwicklung erfolgte nach vorheriger Zusammenstellung der Anforderung und Programmdefinitionen in JAVA8. Dabei wurde der Programmcode mit Hilfe der Versionierungssoftware Git gepflegt und auf der Plattform GitHub in Zusammenarbeit mit der Softwarefirma entwickelt. Die Editierung der Software erfolgte in der Eclipse-Entwicklungs-umgebung für Java. Die Dokumentation der GS wurde im Rahmen des Ausschreibungsverfahrens in Form eines PDF-Dokuments (vgl. 7.1) erstellt. Dort wurden Programmablaufpläne und Algorithmen zur Berechnung der GS festgehalten. Zusätzlich wurden die Grundparameter der GS in Form einer frei zugänglichen Datenbank veröffentlicht. Alle progammiertechnischen Diagramme, die sog. UML-Zeichungen, wurden mit Hilfe des Programmes UML-Designer erstellt.

2.2 Parameter der GS

Vorhandene Daten über die Dauer einzelner Entwicklungsstadien der Geisenheimer Kc-Werte wurden ausgewertet. Betrachtet wurden dafür 817 einzelne Datensätze des Versuchbetriebs Queckbrunnerhof aus dem Zeitraum von 2006 bis 2016, die mit Hilfe eines auf der Programmiersprache R (R Core Team, 2017) basierenden Skriptes verarbeitet wurden. Zur Evaluierung der GS wurden in den Jahren 2017, 2018 und 2019 auf den Flächen des Versuchbetriebs Queckbrunnerhof Zwiebeln kultiviert. Die Veränderung der

Bestandeshöhe, des Wasserverbrauchs der Kultur und der Klimaparameter wurden erfasst. Die Messung der Bestandshöhe erfolgte wöchentlich an randomisiert gewählten Messstellen mit Hilfe eines Gliedermessstabes. Der Wasserverbrauch der Kultur Zwiebel wurde mit Hilfe einer Eddy-Kovarianz-Anlage (LI-7500A Open Path CO_2/H_2O Analyser, Gill New Windmaster Sonic, 7550-220 Smartflux, Fa. Li-Cor, Bad Homburg, Abbildung 1) ermittelt. Diese wurde am Feldrand so aufgestellt, dass die Messsensoren in Hauptwindrichtung über dem Bestand ausgerichtet waren.

Die Auswertung der Messdaten erfolgte mit der Software Eddy-Pro (Fa. Li-Cor, Bad Homburg), wobei die Qualität der Messdaten mit dem Energy-Balance-Closure-Verfahren und den durch Eddy-Pro bereitgestellte Qualitätsmerkmale bewertet wurde. Es wurden nur Messdaten verwendet, die die korrekte Windrichtung aufwiesen, die also von der Fläche des untersuchten Zwiebelbestands kamen. Luftfeuchte, Lufttemperatur, Windgeschwindigkeit, photosynthetisch aktive Strahlung (PAR), Nettoeinstrahlung, Bodenwärmestrom und volumetrischer Bodenwassergehalt zeichneten die sog. BioMet-Sensoren des Eddy-Kovarianz-Messystems sowie die Wetterstation Schifferstadt des DLR-RP auf. Aus den ausgewählten Daten wurde der aktuelle tägliche Kc-Wert der Zwiebelkultur als Quotient von gemessener Verdunstung und Referenzverdunstung gebildet und mit der Geisenheimer Kc-Wert-Empfehlung der jeweiligen Entwicklungsstufe verglichen. Damit ist es möglich, eine qualitative und quantitative Aussage über die Empfehlungen der GS für Zwiebel zu tätigen und mögliche Rückschlüsse auf etwaig erforderliche Anpassungen und auf weitere Fragestellungen zu ziehen. Die Aufbereitung und Auswertung der Daten erfolgte mit der statistischen Programmiersprache R.

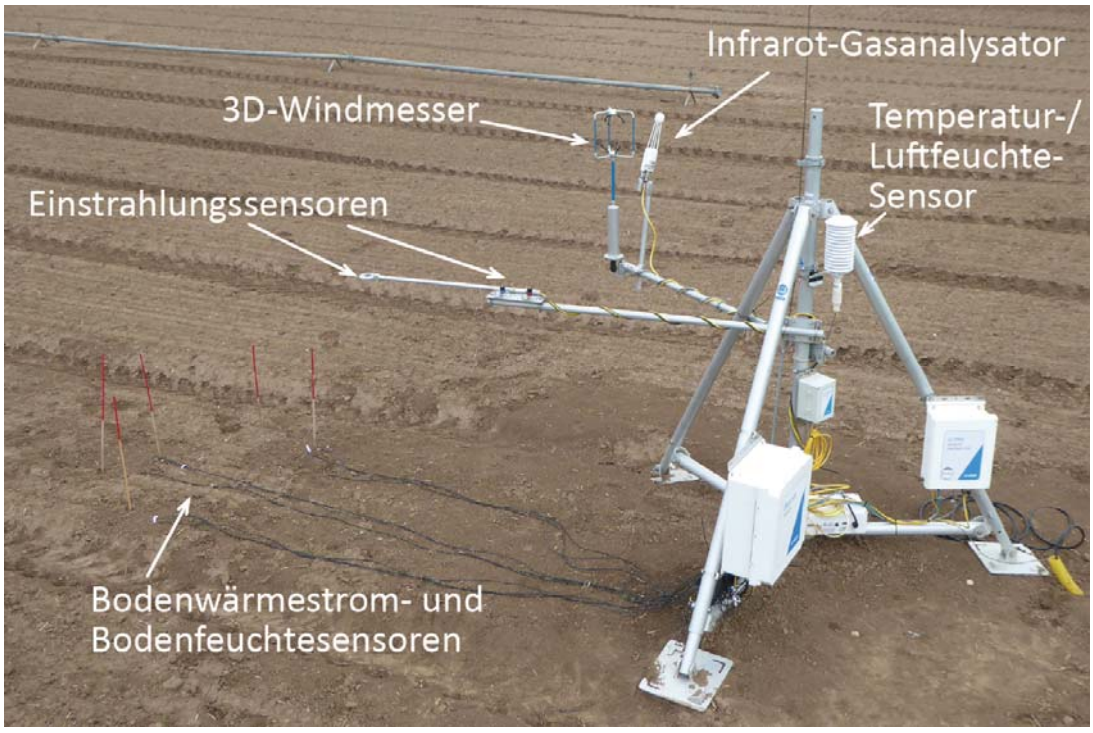

Abbildung 1: Sensoren einer Eddy-Kovarianz-Anlage

3 Ergebnisse

Im Rahmen des Projektes wurde die Bewässerungssoftware GSEHEN erfolgreich entwickelt. Sie wird Nutzern auf der Website der Hochschule Geisenheim zur Verfügung stehen (https://www.hs-geisenheim.de/forschung/institute/gemuesebau/ueberblick-institut-fuer-gemuesebau/bewaesserung/geisenheimer-bewaesserungssteuerung/). Der Quellcode des Projektes ist unter GitHub für Entwickler zugänglich (https://github.com/Gemuesebau/GSEHEN).

Die Messung der Verdunstung von Zwiebel mit dem Verfahren Eddy-Kovarianz war über alle Jahre erfolgreich und wissenschaftlich auswertbar. Dabei konnten wichtige Erkenntnisse zur Entwicklung der Kc-Werte gesammelt werden. Die Geisenheimer Kc-Werte der Modellkultur Zwiebel stellten sich als nicht zu hoch heraus. Vielmehr unterschätzen die Geisenheimer Kc-Werte die tatsächliche Verdunstung zu Beginn der Zwiebelkultur.

3.1 Ausführliche Darstellung der wichtigsten Ergebnisse

3.1.1 AP1: Zusammenfassung der für die Automatisierung von GS benötigten Parameter

Für die Automatisierung der GS sollten vorhandene Daten der HGU und des DLR-RP gesichtet und evaluiert, inkonsistente Daten identifiziert und die verwertbaren Daten für die computergestützte Umsetzung der GS aufbereitet werden. Für die Dauer von Entwicklungsstadien liegen der HGU Datensätze aus Schifferstadt aus dem Zeitraum von 2006 bis 2016 vor. Insgesamt enthalten diese 817 Excel-Tabellenblätter Daten zu erfolgten Bewässerungsvorgängen und die dazu gehörigen Umstelltermine der Kc-Entwicklungsstadien. Aus diesen Daten wurden die Dauern einzelner kc-Entwicklungsstadien mit der Hilfe von individuell erstellten R-Skripten berechnet und nach Plausibilität und Relevanz gefiltert. Eliminiert wurden teils unvollständige oder fehlerhafte Datensätze sowie Datensätze mit nicht repräsentativen Kulturverläufen.

In Abbildung 2 sind die mittleren Entwicklungsdauern der kc-Entwicklungsstadien von 16 Kulturen dargestellt. Beobachtete größere Streuungen sind durch die unterschiedlichen Anbauzeiträume zu erklären. Abbildung 3 zeigt dies am Beispiel von Kopfsalat, die die Daten nach Monaten gruppiert darstellt.

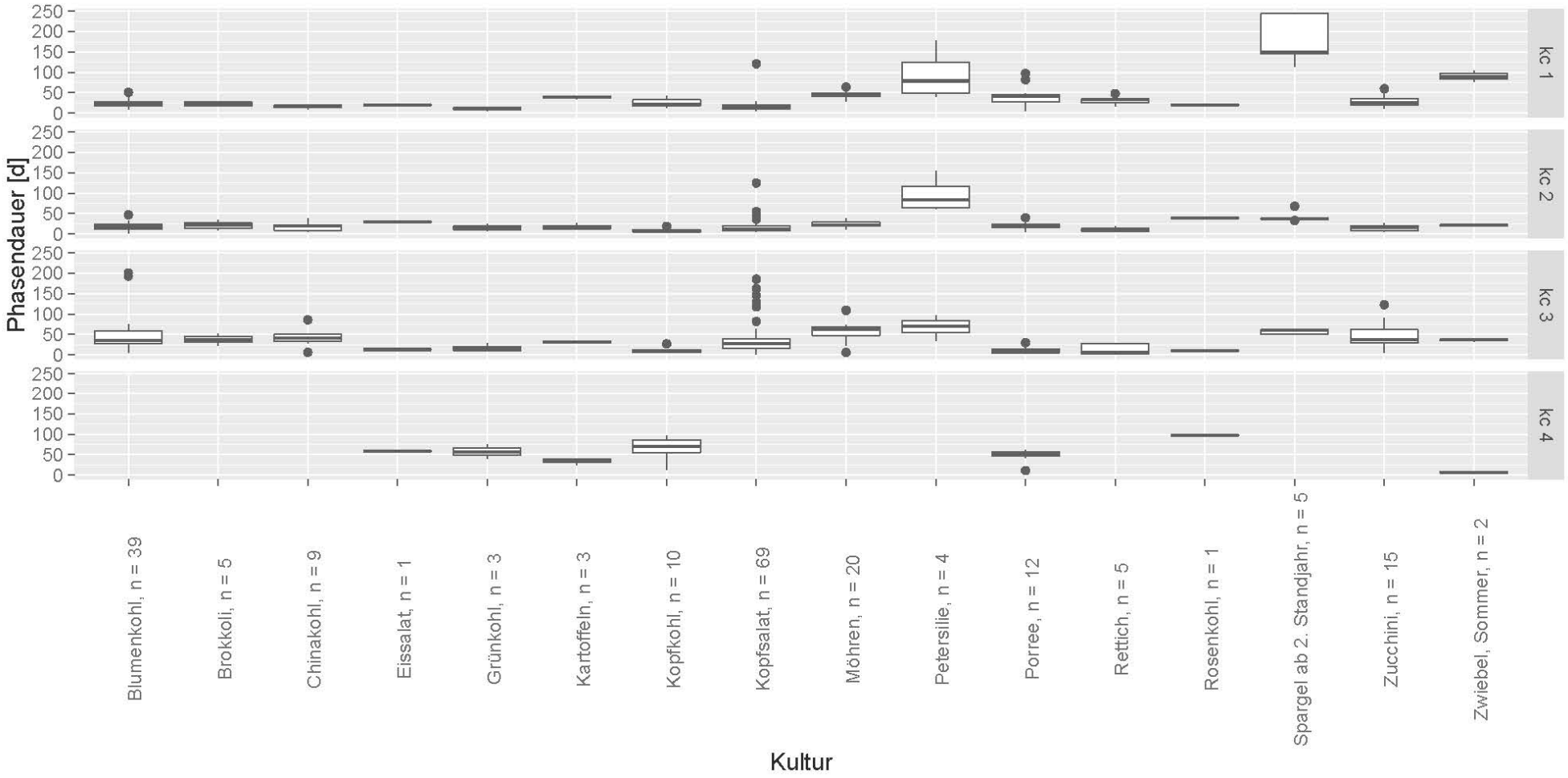

Abbildung 2: Die Dauer der kc-Entwicklungsstadien verschiedener Gemüsekulturen in Tagen (n: Stichprobenumfang).

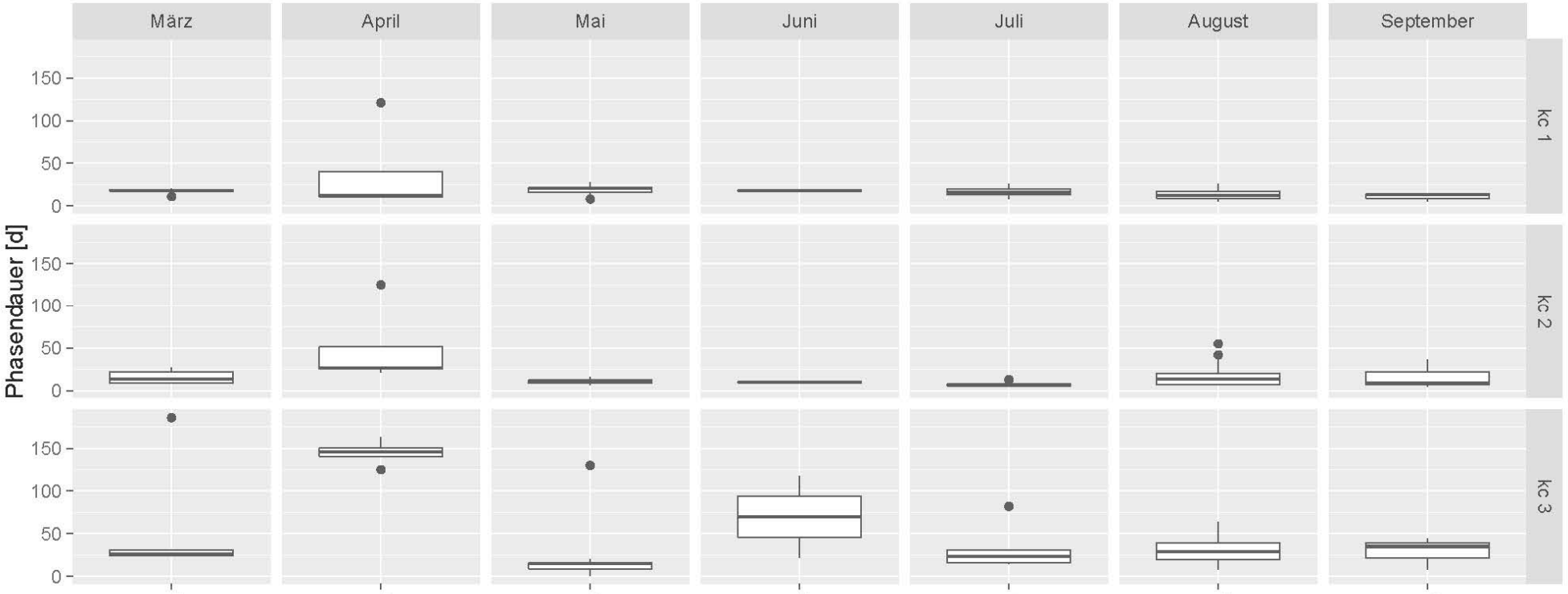

Abbildung 3: Die Dauer der Kc- Entwicklungsstadien 1, 2 und 3 von Kopfsalat in Abhängigkeit des Monats des Kulturbeginns in Tagen

3.1.2 AP2: Evaluierung inkonsistenter Parameter

Die Geisenheimer Kc-Werte müssen sich immer wieder vielfältiger Kritik stellen. Der bisherige Prozess der Kc-Wert-Bildung umfasste die Parametrisierung der Werte in der Geisenheimer Lysimeteranlage sowie eine anschließende Evaluierung dieser Werte in Freilandversuchen mit dem Ziel der Ertragsoptimierung. Dennoch wurden die Geisenheimer kc-Werte durch Berater und Praktiker häufig als zu hoch angesehen. Diese Einschätzung basiert auf Rückmeldungen der gemüsebaulichen Praxis sowie auf der Einschätzung von Allen et al. (1998), dass Kc-Werte über 1 sehr selten vorkommen. Die Praxis ist oft an vielfältige Rahmenbedingungen gebunden, die den Einsatz von Bewässerungswasser limitieren. Die damit einhergehende Skepsis gegenüber höheren Bewässerungsmengen könnte mitunter auf diesem Umstand beruhen.

Um der Kritik nachzugehen, mit GS entstünde ein zu hohes Wasserangebot, steht im Projekt eine neue Messmethode zur Verfügung, um den tatsächlichen Wasserverbrauch unter Feldbedingungen messen zu können, die sogenannte Eddy-Kovarianz-Messung. Bei diesem Messverfahren wird die tatsächliche aktuelle Verdunstung des Pflanzenbestandes über die Erfassung des Wasserdampf-Flusses gemessen, indem die vertikale Luftbewegung in Beziehung (Kovarianz) zum Wasserdampfgehalt der Luft gesetzt wird. Die Luftbewegungen über einem Pflanzenbestand erfolgen in Wirbeln, den sogenannten Eddys. Die hohe Geschwindigkeit dieser Luftwirbel erfordert eine Messtechnik für Windgeschwindigkeit und Wasserdampfgehalt mit einer Auflösung einer Zehntelsekunde. Als wichtigste Voraussetzung einer erfolgreichen Eddy-Kovarianz-Messung gilt die Auswahl einer geeigneten Fläche. Die vorherrschende Windrichtung ist entscheidend dafür, ob die Messwerte von der zu untersuchenden Fläche stammen. Der Messbereich der Eddy-Kovarianz-Messung ist i.d.R. 100 bis 200 Metern von der Station entfernt und stellt ein lang gezogenes Oval dar. Der Messbereich rückt weiter weg, je höher sich das Messgerät über der Bestandsoberfläche befindet und rückt während des Wachstums von Pflanzenbeständen näher, weil sich der Abstand Messgerät zu Pflanzenoberfläche verringert. Ein zu geringer Abstand von Messgerät und Pflanzenbestand ist zu vermeiden, da Eddys nahe an der Bestandesoberfläche zunehmend kleiner und schneller werden und dann die maximale Auflösung der Messtechnik überschritten wird. Über den Kulturverlauf ist die Höhe des Pflanzenbestandes zu erfassen. Abbildung 4 zeigt die regelmäßig gemessene Pflanzenhöhe am Beispiel des Versuchsjahrs 2018.

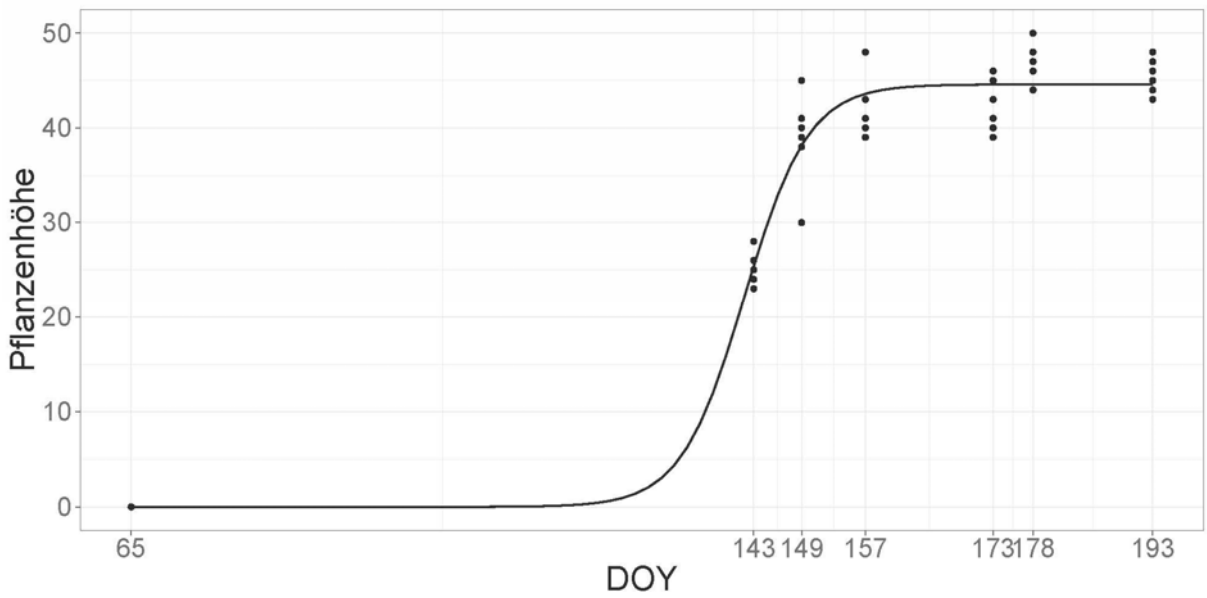

Abbildung 4: Entwicklung der Pflanzenhöhe über die Kalendertage [DOY] des Zwiebelbestandes 2018 mit eingefügtem Wachstumsmodell, das als sog. Metadaten in die Berechnung der Eddy-Kovarianz einging

Kommt der Wind auf der Fläche aus ständig wechselnden Richtungen, ist der optimale Messpunkt stets inmitten eines Feldes, der im Umkreis von über dreihundert Metern von dem einheitlichen Pflanzenbestand umschlossen ist. Im Umfeld einer Eddy-Kovarianz-Messung dürfen keine Hügel oder großen Objekte stehen. Diese würden die Windverhältnisse so verändern, dass Stoffflüsse nicht mehr komplett erfasst werden können und die Messwerte unbrauchbar wären. Windverhältnisse (Abbildung 5) und Bebauung in Geisenheim lassen einen Einsatz auf den eigenen Versuchsflächen nicht zu. Die Flächen des Versuchsbetrieb Queckbrunnerhof des DLR-RP erfüllen hingegen alle notwendigen Voraussetzungen und weisen eine ausgeprägte Hauptwindrichtung auf (Abbildung 6).

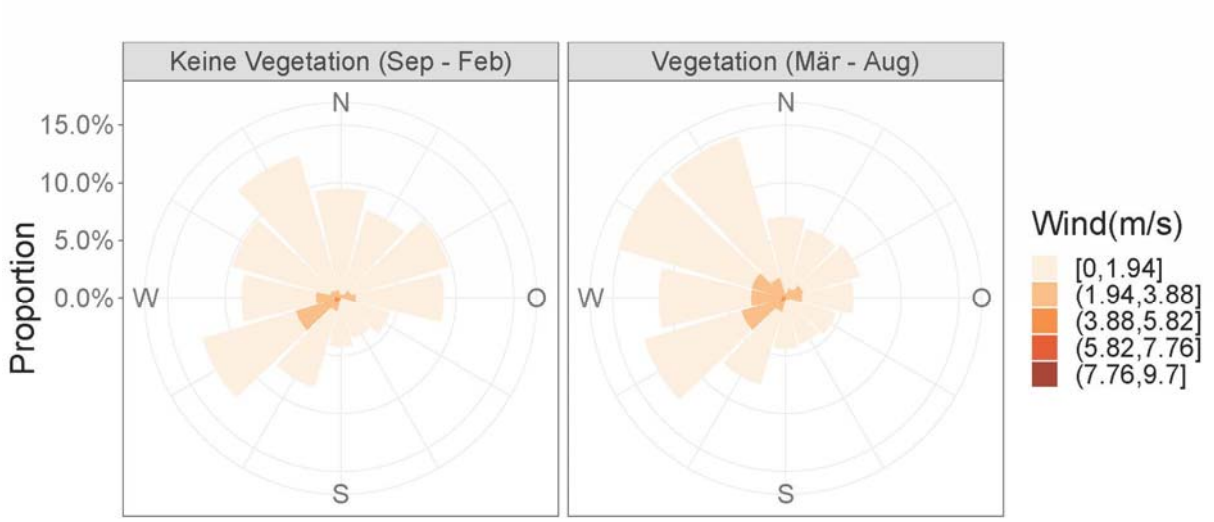

Abbildung 5: Windgeschwindigkeit und Windrichtung innerhalb und außerhalb der Vegetationsperiode der Jahre 2007-2015 für den Standort **Geisenheim**

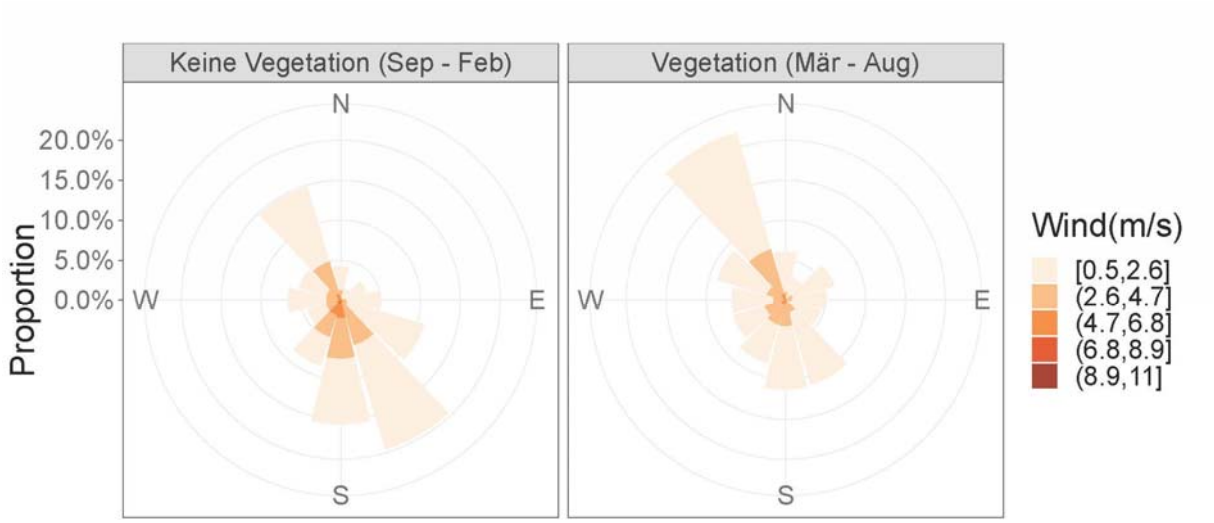

Abbildung 6: Windgeschwindigkeit und Windrichtung innerhalb und außerhalb der Vegetationsperiode der Jahre 2011-2016 für **Schifferstadt**

Als Maß für die Qualität der Eddy-Kovarianz-Daten wird die sog. Energy-Balance-Closure-herangezogen, welche auf dem ersten Hauptsatz der Thermodynamik basiert: Die Energie in einem abgeschlossenen System ist konstant. Für die Eddy-Kovarianz gilt: Die Summe von latenter und sensibler Wärme (LE + H), die mit der Eddy-Kovarianz gemessen werden, muss gleich der Differenz von globaler Nettoeinstrahlung und dem Bodenwärmemstrom (Rn - G) sein, die mit Referenzsensoren erfasst werden. In einem idealen System läge diese Beziehung bei eins. Qualitativ hochwertige Eddy-

Kovarianz-Messungen in einem realen System weisen einen Faktor über 0,8 auf. In allen drei Versuchsjahren lag der Faktor über 0,82 in einem publikationswürdigen Bereich (Abbildung 7).

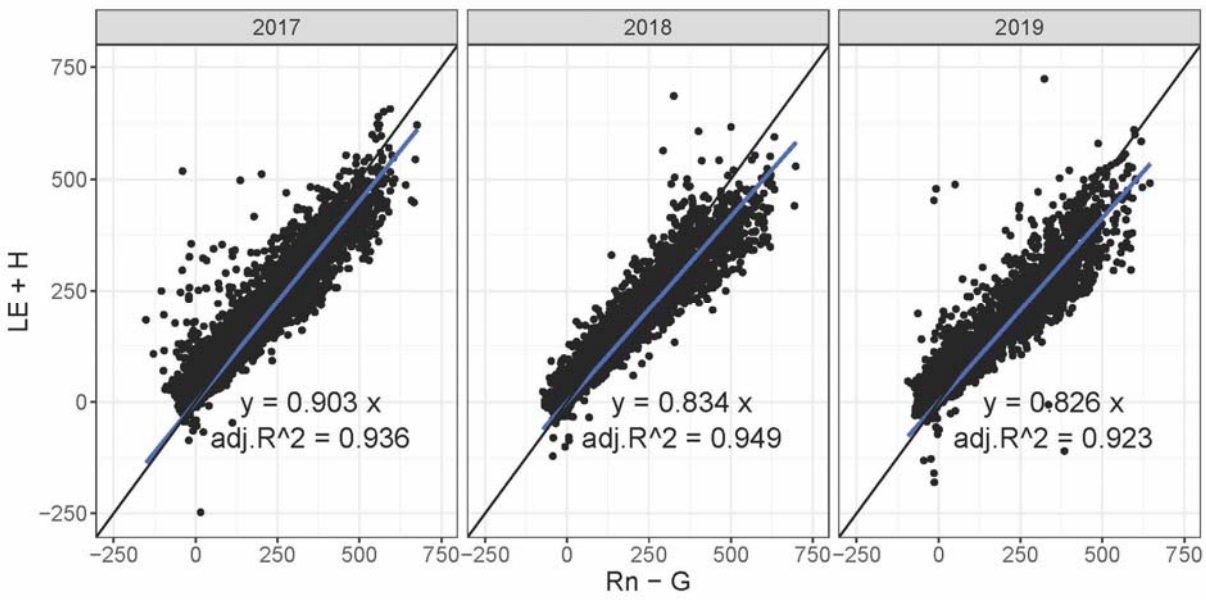

Abbildung 7: ’Energy Balance Closure’ der Eddy-Kovarianz-Messung in Schifferstadt für die Jahre 2017, 2018 und 2019:
Vergleich der vom Eddy-Kovarianz-System gemessene Energiebilanz (latente Wärme LE + sensible Wärme H)) mit der aus Referenzsensoren (globale Nettoeinstrahlung Rn - Bodenwäremstrom G)
Steigung > 0,8 = hohe Qualität der Messdaten

Um die Qualität der Geisenheimer Kc-Werte bewerten zu können, wurde der tatsächliche Kc-Wert der Kultur gebildet: aus dem Quotienten der mit der Eddy-Kovarianz gemessenen Verdunstung und der aus Klimadaten berechneten Referenzverdunstung. Abbildung 8 bis Abbildung 10 zeigen für die Jahre 2017 bis 2019 den tatsächlich gemessenen Kc-Wert ($Kc_{Gemessen}$), dessen gleitendes Mittel ($Kc_{Gleitend}$), und die für die Entwicklungsstadien der Zwiebel gemittelten Kc-Werte ($Kc_{Stadien}$) sowie die für die Geisenheimer Steuerung empfohlenen Kc-Werte (Kc_{GS}).

Die drei Versuchsjahre zeigen sehr ähnliche Ergebnisse. In allen drei Jahren liegen die gemessenen kc-Werte im ersten Entwicklungsstadium "ab Auflaufen" mit Kc 1,11 bis 1,19 deutlich über der Empfehlung von Kc 0,7 (Tabelle 1). Die Geisenheimer Empfehlung unterschätzt also die tatsächliche Verdunstung im ersten Entwicklungsstadium. Im Jahr 2017 und 2018 liegt auch der gemessene kc-Wert für das zweite Entwicklungsstadium "ab 5. Blatt" mit Kc 1,46 bis 1,58 noch geringfügig über der Empfehlung von kc 1,3. Im dritten Entwicklungsstadium "ab 8. Blatt" liegen die gemessenen Kc-Werte der drei Versuchsjahre mit kc 1,52 bis kc 1,75 im Bereich der Empfehlung von kc 1,6.

Ob die anfängliche Unterschätzung der Evapotranspiration als kritisch einzuordnen ist, bedarf weiterer Untersuchungen. Ein zu Kulturbeginn reduziertes Wasserangebot könnte sich pflanzenbaulich auch positiv auswirken, weil das Risiko bewässerungsbedingten Verschlämmens des Bodens reduziert oder das Wurzelwachstum der Zwiebel durch verringerte Wassergaben angeregt wird.

Eine genauere Abschätzung des Kc-Wertes nach der Etablierung der Kultur könnte durch komplexere Modelle möglich sein, die den Einfluss der Bodenbeschaffenheit stärker einbeziehen. Für eine praktische Anwendung der GS sind die ermittelten Kc-Werte jedoch ausreichend präzise und führen zu pflanzenbaulich und arbeitswirtschaftlich sinnvollen Bewässerungsempfehlungen.

Mit den Untersuchungen in diesem Projekt ist der Nachweis erbracht, dass die Geisenheimer Kc-Werte für Zwiebel nicht zu hoch sind.

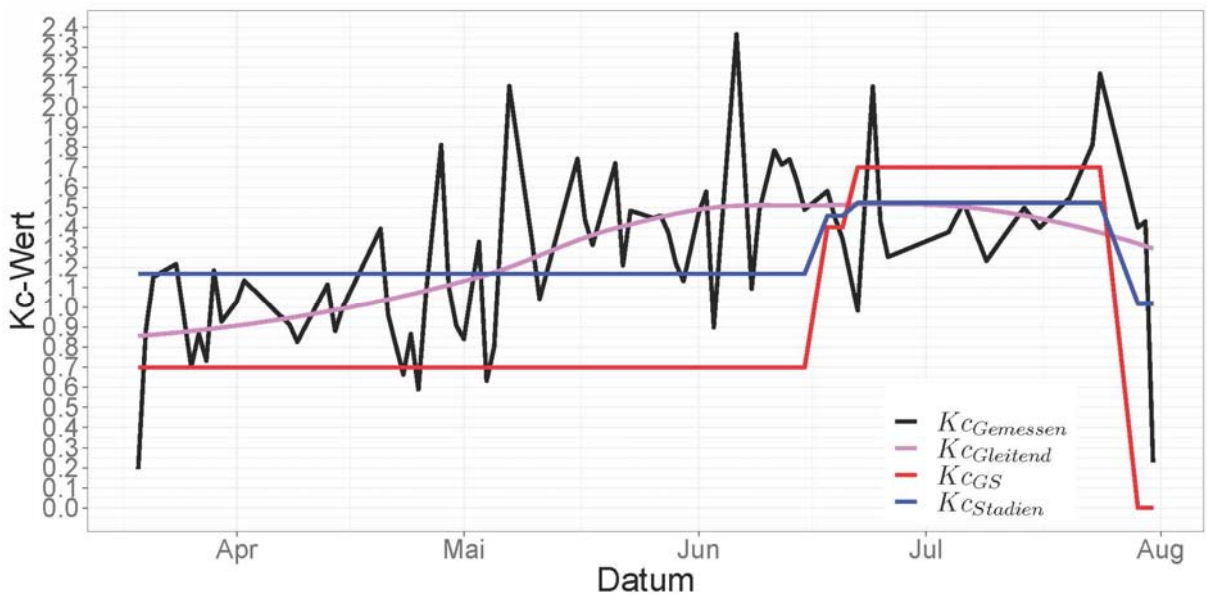

Abbildung 8: Zwiebeln 2017: zeitliche Veränderung des gemessenen Kc-Werts ($Kc_{Gemessen}$), des gleitenden Mittels ($Kc_{Gleitend}$), der für die Entwicklungsstadien gemittelten Messwerte ($Kc_{Stadien}$), sowie der von GS empfohlenen Kc-Werte (Kc_{GS}).

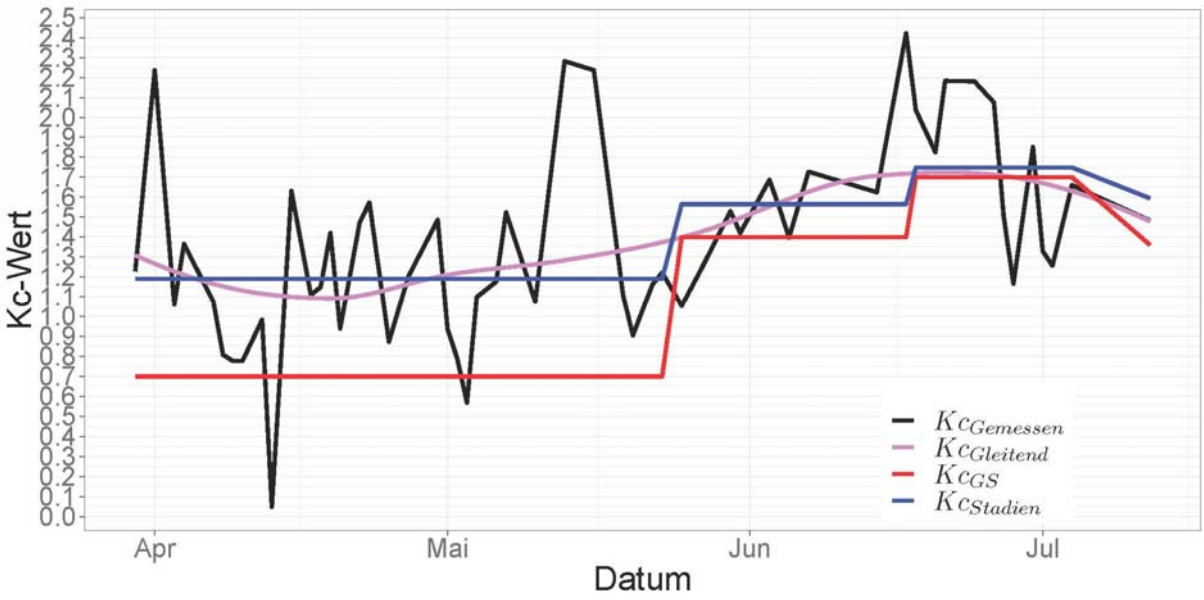

Abbildung 9: Zwiebeln 2018: zeitliche Veränderung des gemessenen Kc-Werts ($Kc_{Gemessen}$), des gleitenden Mittels ($Kc_{Gleitend}$), der für die Entwicklungsstadien gemittelten Messwerte ($Kc_{Stadien}$), sowie der von GS empfohlenen Kc-Werte (Kc_{GS}).

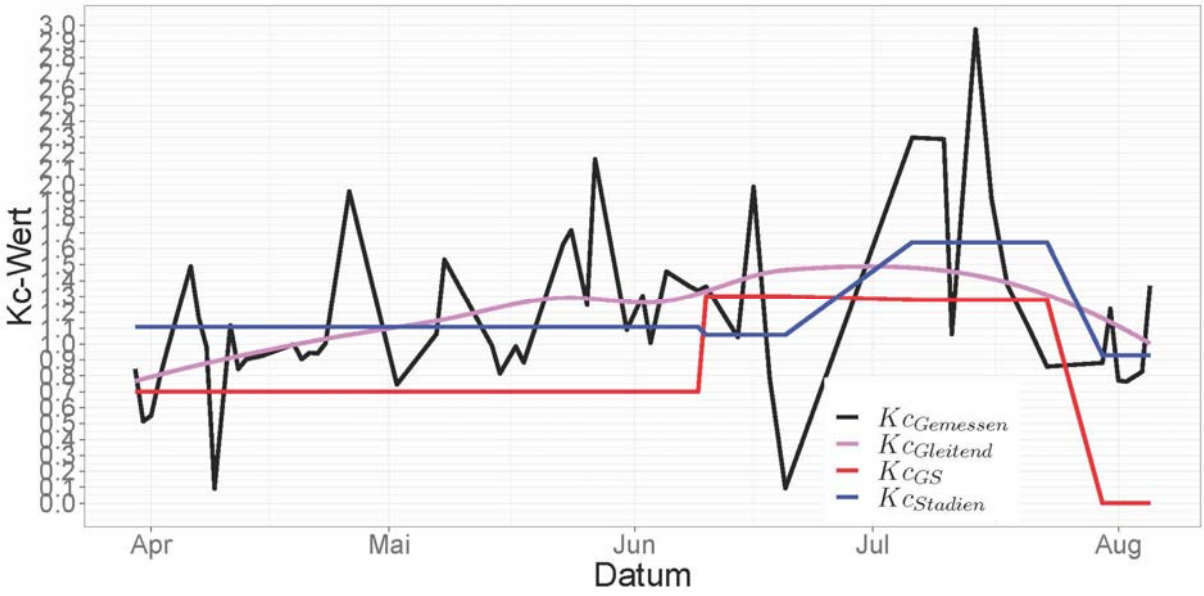

Abbildung 10: Zwiebeln 2019: zeitliche Veränderung des gemessenen Kc-Werts ($Kc_{Gemessen}$), des gleitenden Mittels ($Kc_{Gleitend}$), der für die Entwicklungsstadien gemittelten Messwerte ($Kc_{Stadien}$), sowie der von GS empfohlenen Kc-Werte (Kc_{GS}).

Tabelle 1: Für Zwiebel empfohlenen Kc-Werte (Kc_{GS}) im Vergleich zu Kc-Werten, die aus Eddy-Kovarianz-Messungen ($Kc_{Stadien}$) abgeleitet wurden

Entwicklungsstadium	Kc_{GS}	$Kc_{Stadien}$		
		2017	2018	2019
1. ab Auflaufen	0,7	1,17	1,19	1,11
2. ab 5. Blatt	1,3	1,46	1,56	NA
3. ab 8. Blatt	1,6	1,52	1,75	1,64

3.1.3 AP3: Dokumentation von GS

Die Dokumentation der GS ist im Rahmen der Ausschreibung erstellt und dem Anhang 7.1 zu entnehmen.

3.1.4 AP4: Konzipierung der Datenbank

Für die Speicherung und Bereitstellung der Grunddaten der GS wurden verschiedene Konzepte einer PostgreSQL-Datenbank geprüft. Es wurde eine PostgreSQL-Datenbank gemietet, die als Cloud-Anwendung durch die Firma Tec-Racer zur Verfügung gestellt wird.

3.1.5 AP5: Ausschreibung der Unterauftragsvergabe durch HGU

Das Leistungsverzeichnis und Anforderungsprofil wurde erstellt. Nach einem erfolgreich durchgeführten Ausschreibungsverfahren wurde der Auftrag an die Firma CGS mBH (Braunschweig) vergeben. Mit der Softwareentwicklung konnte zum 01.12.2017 begonnen werden. Der Ausschreibungstext und das Leistungsverzeichnis sind dem Anhang 7.1 zu entnehmen.

3.1.6 AP6: Umsetzung der Datenbank

Im Rahmen des Projektes sollte nicht nur die Umsetzung der GS in eine Software ermöglicht werden, sondern auch die Grunddaten der Geisenheimer Steuerung in einer Form veröffentlicht werden, welche die informationstechnische Weiterverarbeitung in ähnlichen Systemen ermöglicht. Dies sollte so erfolgen, dass die HGU für die Aktualität und Validität dieser Grunddaten immer die letzte Verfügungsgewalt besitzt. Dazu wurden die Daten in Form einer Datenbank publiziert. Nach der Konzipierung und Einrichtung der Datenbank wurden die Kulturparameter der GS für 27 Kulturen der Datenbank hinzugefügt und mit Werten zur Dauer der Kc-Entwicklungsstadien ergänzt. Die Software GSEHEN ermöglicht nun die Berechnung der Bewässerungsbedürftigkeit für 27 Kulturen sowie den unkomplizierten Transfer der Funktionalität an weitere Interessierte. Die für die Verwendung in der Applikation vorgesehenen Werte sind dem Anhang 7.2 zu entnehmen.

3.1.7 AP7: Umsetzung des KWB-Moduls durch HGU

Zur Berechnung der Klimatischen Wasserbilanz nach GS innerhalb der GSEHEN-Software wurden die Berechnungsalgorithmen in Java umgesetzt und getestet. Dabei wurden die Funktionalitäten integriert, die die Ausschreibung forderte. Dazu zählen die Einführung eines betriebsspezifischen Skalierungsfaktors für die Bewässerung sowie die freie Konfigurierbarkeit von Bewässerungspausen in der Bilanzierung nach Starkregen-ereignissen. Der Quellcode ist dem Anhang 7.3 zu entnehmen.

3.1.8 AP8: Umsetzung des Wetterdatenmoduls durch Unterauftragnehmer

Das Wetterdatenmodul ist vom Unterauftragnehmer umgesetzt. In der derzeitigen Programmversion ist der Import von Wetterdaten in einer vorgegebenen Datenstruktur möglich.

3.1.9 AP9: Umsetzung des GUI-Moduls durch Unterauftragnehmer

Im Rahmen der Umsetzung des GUI-Moduls wurden in Zusammenarbeit zwischen HGU und CGS die Benutzeroberfläche des GSEHEN-Programmes entwickelt. Der Funktionsumfang der Benutzeroberfläche orientiert sich dabei an den durch die Ausschreibung definierten Aufgaben, die der Nutzer mit dem Programm bearbeiten kann.

Dazu zählen:

- betriebsspezifische Parameter anpassen
- Felder anlegen und konfigurieren
- Schläge und Anbausätze anlegen und konfigurieren
- Bewässerungsgrenzwerte konfigurieren
- Startpunkt der Bewässerungsberechnung wählen
- aktuellen Bewässerungsbedarf der Kulturen ermitteln und anzeigen
- Bewässerungsmengen buchen
- Pflanzenentwicklungsstatus buchen
- Dokumentation und Archivierung der Bewässerungsmaßnahmen

Die einzelnen Aufgaben wurden in die graphische Benutzeroberfläche integriert und werden in den nachfolgenden Ansichten dargestellt.

3.1.10 Startansicht

Die Startansicht von GSEHEN (Abbildung 11) ermöglicht das Anlegen des Betriebes (Abbildung 12), der Felder (Abbildung 13) und der Schläge (Abbildung 14), unterstützt von einer Kartenansicht. Des Weiteren werden Detailinformationen zu den Schlägen gegeben. Abbildung 15 stellt dar, wie der Bewässerungsbedarf eines Schlages angezeigt wird.

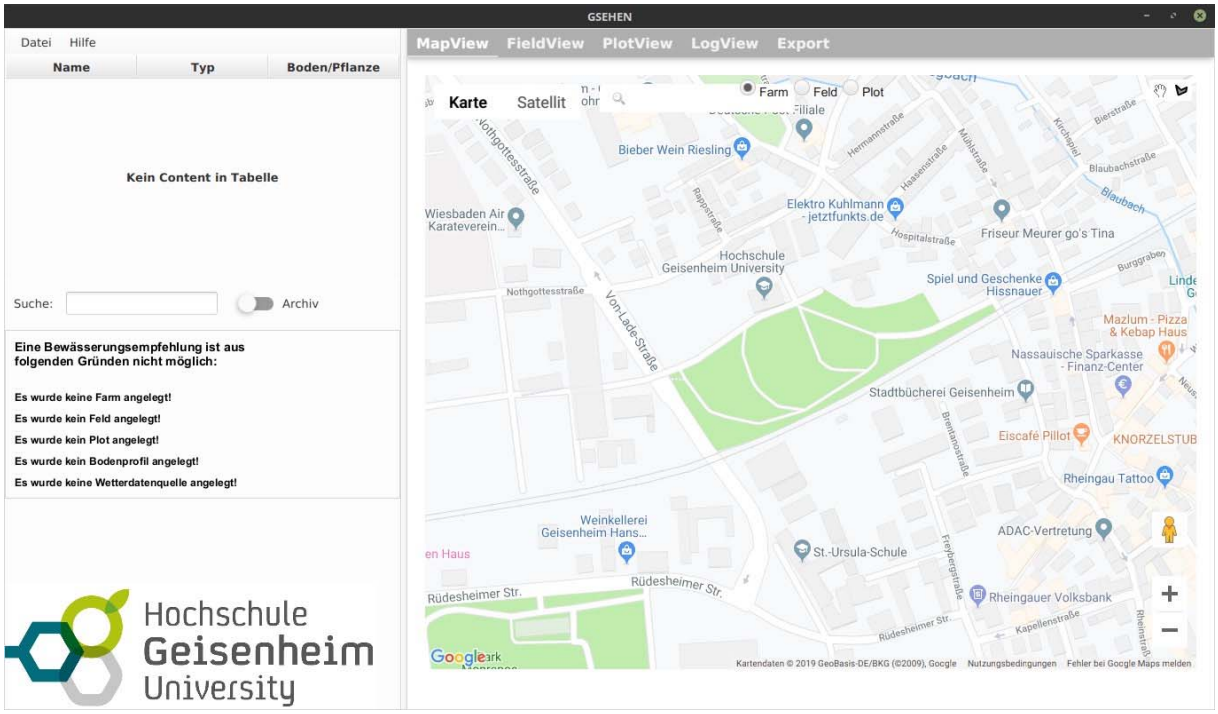

Abbildung 11: Startansicht von GSEHEN

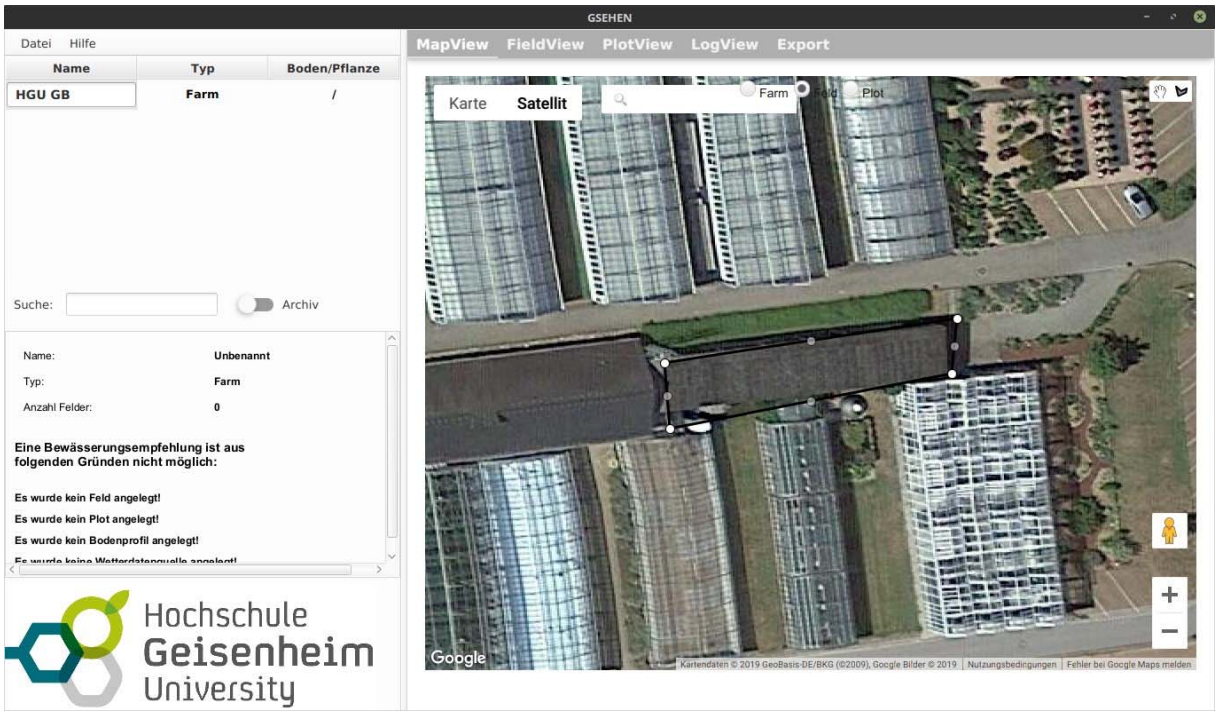

Abbildung 12: Das Anlegen eines Betriebs; schwarzes Polygon = Betrieb

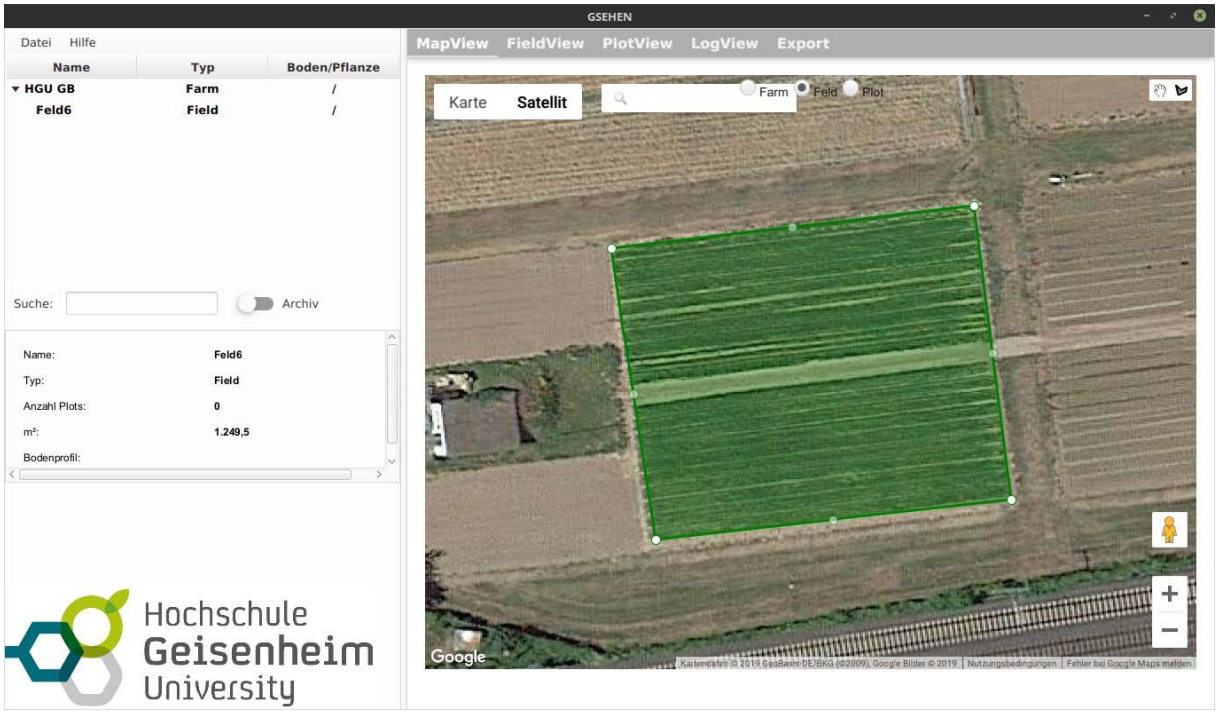

Abbildung 13: Das Anlegen eines Feldes; grünes Polygon = Feld

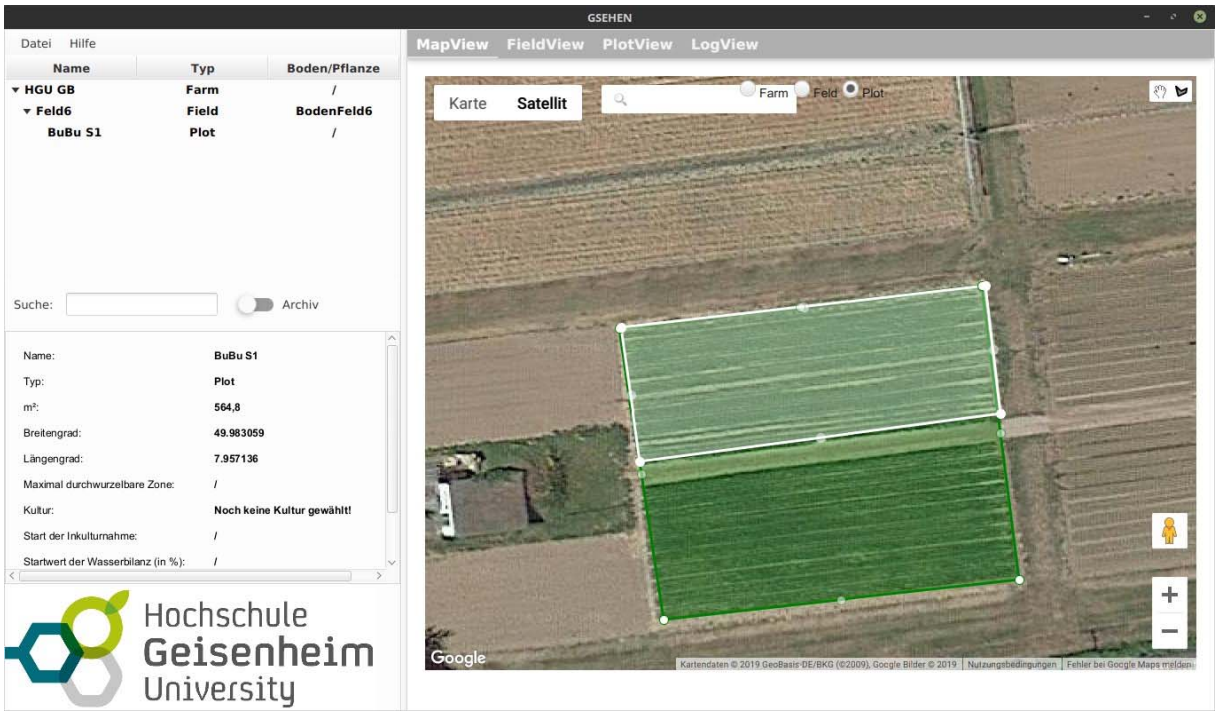

Abbildung 14: Das Anlegen eines Schlages; weißes Polygon = Schlag

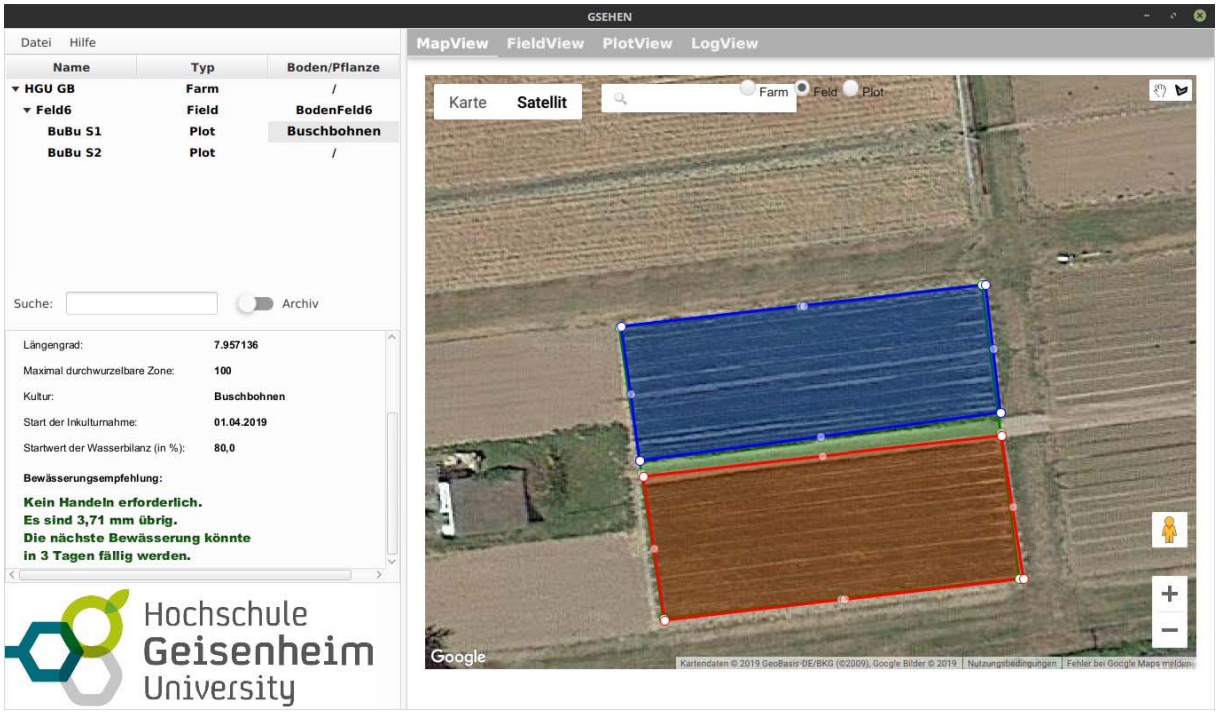

Abbildung 15: Übersicht über die Bewässerungswürdigkeit der Schläge:
blau = kein Handeln erforderlich
rot = Bewässerung notwendig
Detaillierte Angaben links mit grünem Beschreibungstext

3.1.11 Feldansicht

Über die Feldansicht (Abbildung 16) werden Wetterdatenquellen und Bodenprofile für die in der Startansicht angelegten Felder konfiguriert. Dazu stehen für Bodenprofile und Wetterdatenquellen jeweils eigene Konfigurationsansichten bereit (Abbildung 17 und Abbildung 18).

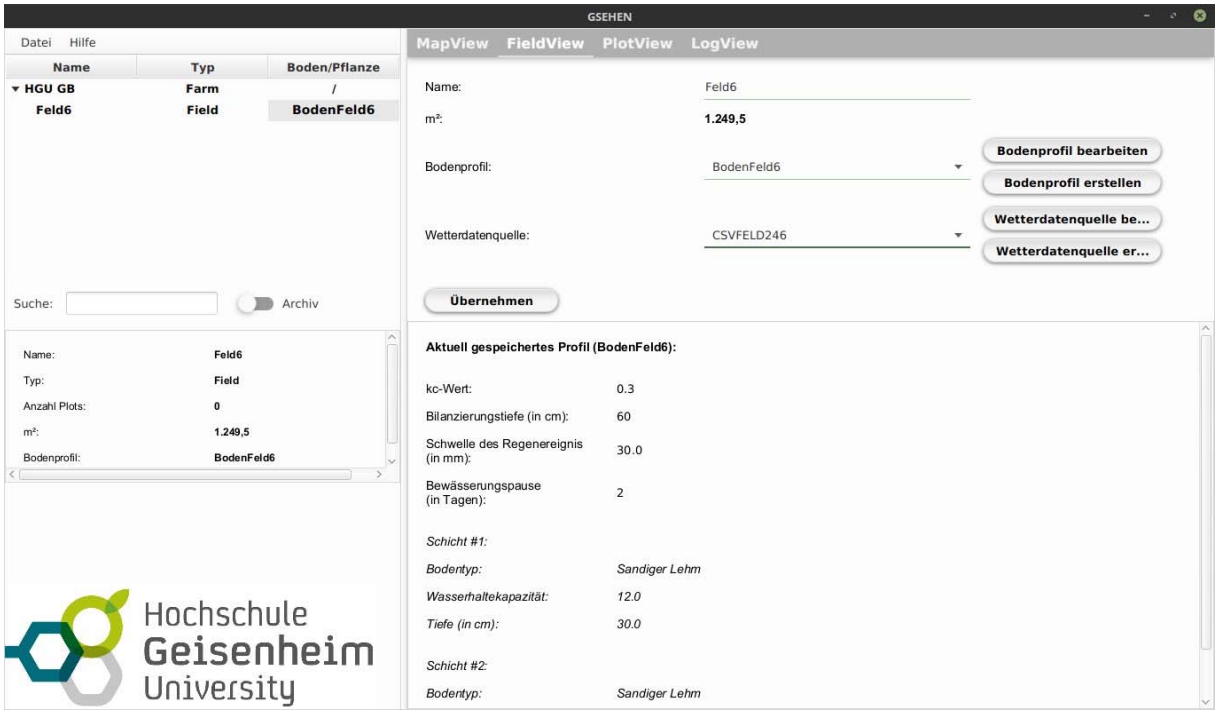

Abbildung 16: Feldansicht von GSEHEN

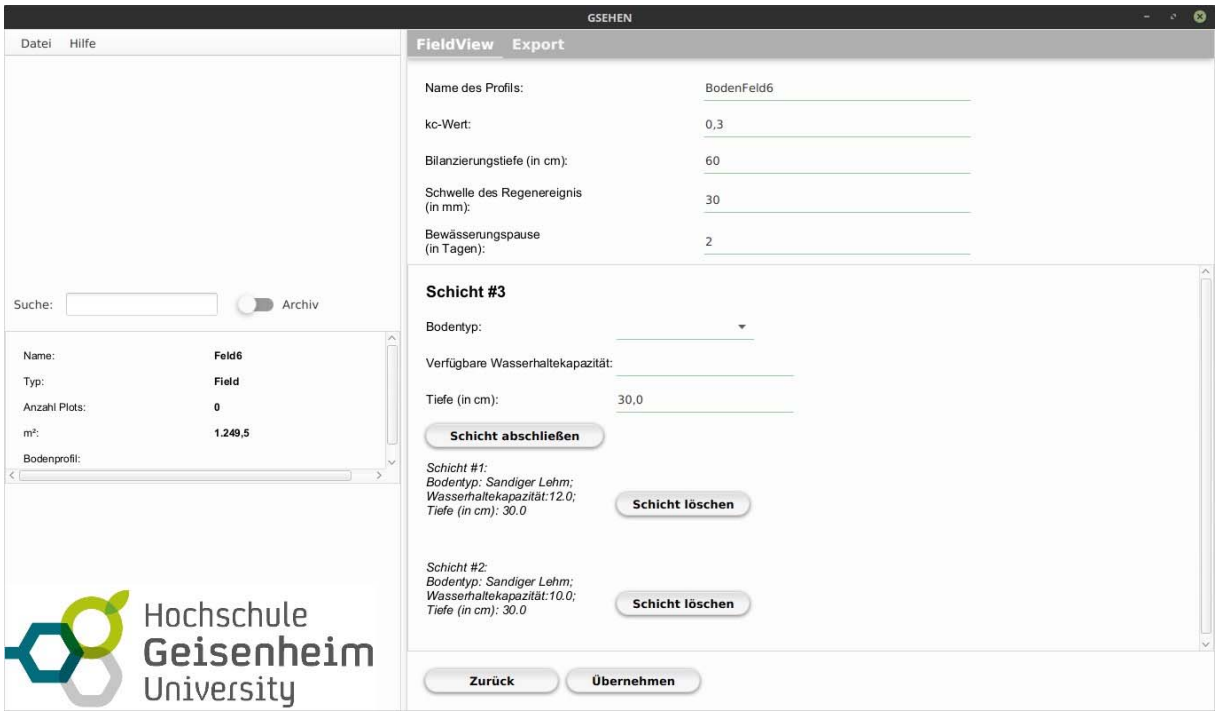

Abbildung 17: Konfiguration des Bodenprofils mit einzelnen Schichten und Bodenarten

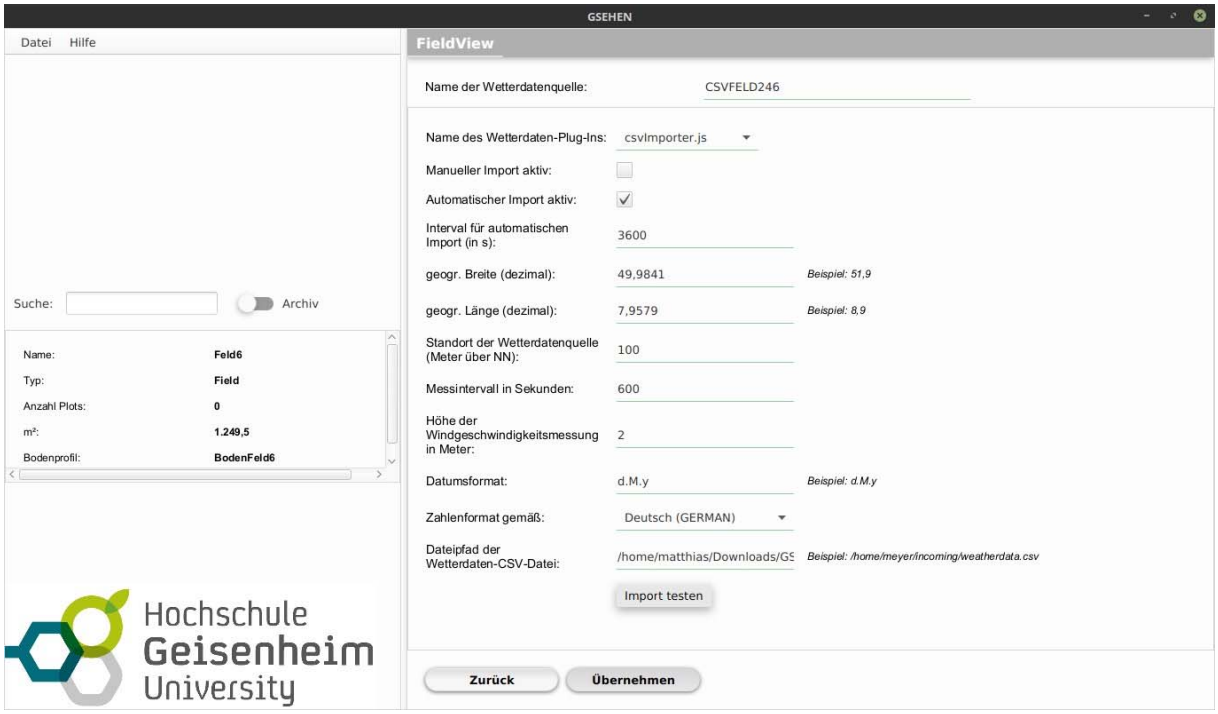

Abbildung 18: Konfiguration der Wetterdatenquelle

3.1.12 Schlagansicht

In der Schlagansicht (Abbildung 19) werden der Schlag mit seinen Kulturparametern konfiguriert. Dazu zählen:

- Maximal durchwurzelbare Bodentiefe
- Zeitpunkt und der Art des Starts der Bilanzierung:
  Verzögerter Start mit anfänglicher Bilanzierung der Evaporation des unbewachsenen Bodens oder
  sofortiger Start mit Bilanzierung der Evapotranspiration der Kultur
- Startwert der Wasserbilanz
- Verwaltung der im Schlag angebauten Kultur mit den Übergängen und Dauern der Entwicklungsstadien und dem Kulturabschluss durch die Ernte
- Buchung von Bewässerungsereignissen und Niederschlag

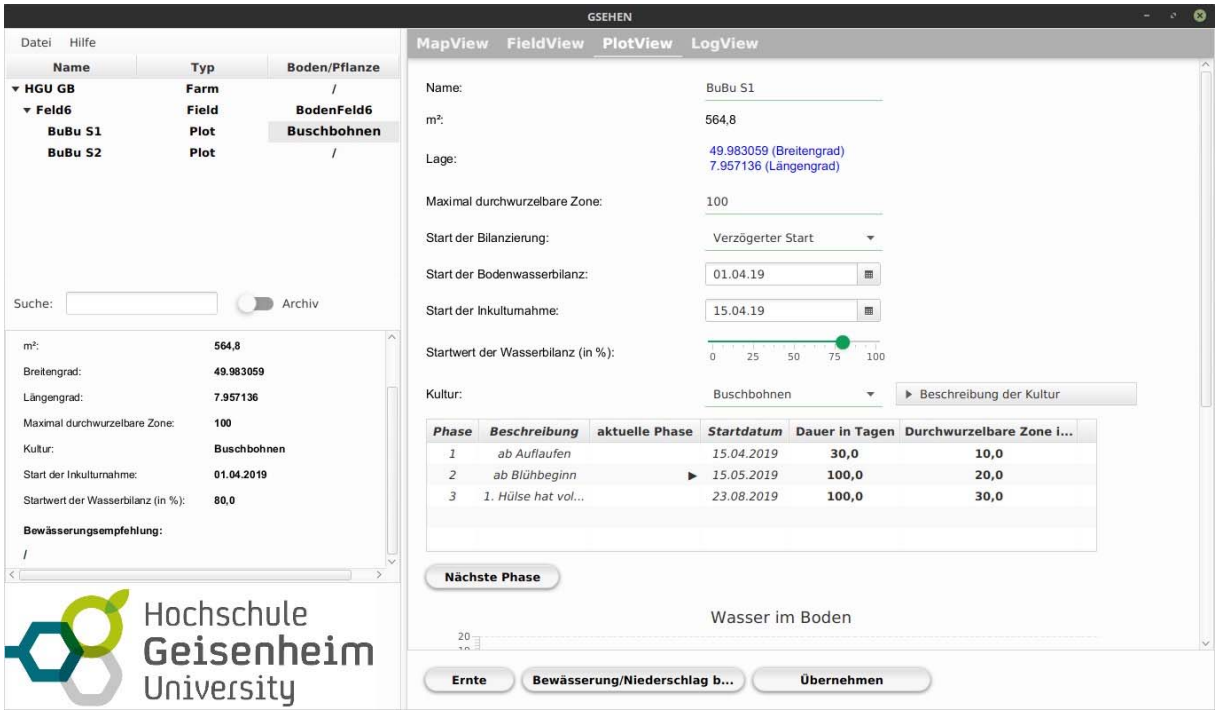

Abbildung 19: Schlagansicht: Konfiguration der Kulturparameter zur Berechnung der GS

3.1.13 AP10: Testdurchlauf des Entscheidungshilfesystems

Ein Prototyp der GSEHEN-Software wurde an der HGU im Rahmen eines Buschbohnen-Versuches von Ende Juli bis Mitte Oktober 2019 erfolgreich zur Steuerung der Bewässerung eingesetzt.

3.2 Voraussichtlicher Nutzen und Verwertbarkeit der Ergebnisse

Die im Projekt entwickelte Software wird allen interessierten Nutzern aus der Gemüsebau-Praxis und -Forschung als Unterstützungstool für die Bewässerungsplanung zur Verfügung stehen. Damit trägt GSEHEN zu einer nachhaltigen Nutzung der Ressource Wasser bei und verringert Überbewässerung und Nitratauswaschung.

Des Weiteren kann die Software durch ihre Dokumentation und quelloffene Entwicklung von Unternehmen der Agrarsoftware-Branche dazu genutzt werden, die Geisenheimer Steuerung fachgerecht in ihre bestehenden Systeme zu integrieren.

Mit der Bereitstellung der Grunddaten der Geisenheimer Steuerung als Datenbank wird sichergestellt, dass zukünftig für alle Nutzer der Geisenheimer Steuerung aktuelle Daten, in einer datenverarbeitungsfreundlichen Weise zur Verfügung stehen.

Die im Projekt gewonnen Erkenntnisse zur Überprüfung der Kc-Werte bei Zwiebel werden in Fachpublikationen veröffentlicht und leisten einen wertvollen Beitrag zum Verständnis der Kc-Wert Entwicklung im Freiland. Die Versuche bilden eine Grundlage für folgende Fragestellungen und Projekte.

4 Zusammenfassung

Im Projekt GSEHEN wurde zwei Aufgabenstellungen bearbeitet: Zum einen die Entwicklung und Dokumentation der Bewässerungssoftware GSEHEN auf Grundlage der Geisenheimer Steuerung und zum anderen die Überprüfung der Parameter der Geisenheimer Steuerung, der Kc-Werte, mit Hilfe des Messverfahrens Eddy-Kovarianz.

Die Bewässerungssoftware wurde in JAVA als Open Source Software entwickelt und ist samt Dokumentation als Anwendungssoftware und Quellcode verfügbar. Sie ermöglicht eine schlagspezifische Planung, Verwaltung und Dokumentation von Bewässerungsvorgängen auf Grundlage der Geisenheimer Steuerung. Eine moderne Kartenansicht wird dazu genutzt, die eine komfortable Zuordnung von Feldern und Schlägen und eine Bearbeitung von allen notwendigen Benutzereingaben zur Berechnung der Geisenheimer Steuerung ermöglicht und vereinfacht.

Die Überprüfung der Kc-Werte ergab neue Erkenntnisse über deren Verlauf und Ausprägung bei der Modellkultur Zwiebel. So konnte eine anfängliche Unterschätzung des Wasserverbrauches festgestellt werden. Im weiteren Kulturverlauf schätzen die empfohlenen Kc-Werte den Wasserverbrauch jedoch treffend. Die Geisenheimer Kc-Werte tragen somit zu einer bedarfsgerechten Bewässerungsempfehlung bei. Die Erkenntnisse wurden durch den Einsatz der Eddy-Kovarianz gewonnen, die sich für die Messung der Evapotranspiration unter Freilandbedingungen sehr gut eignete.

5 Gegenüberstellung der ursprünglichen geplanten zu den tatsächlich erreichten Zielen und Hinweise auf weiterführende Fragestellungen

Die im Projekt geplanten Ziele konnten erreicht werden. Die Software wurde zu einer benutzbaren Version entwickelt, die den Ansprüchen der Ausschreibung genügt. Die Software steht der Öffentlichkeit zur Verfügung.

Ein ungelöstes Problem bleibt die verlässliche Einbindung von Wetterdaten in das Programm. Jeder Nutzer muss sicherstellen, dass die Daten in einer vom Programm vorgegebenen Struktur vorliegen. Weiterhin gilt: "eine Softwareentwicklung ist nie fertig". Die Übergabe der Software in die Open-Source-Community stellt einen wichtigen Beitrag dazu bei, diesem Umstand gerecht zu werden und das Problem einer Lösung zuzuführen. Lösungsansätze bietet hier möglicherweise eine zukünftige Anbindung an staatlich verfügbare Datenschnittstellen.

Die Überprüfung der Kc-Werte ergab wertvolle Erkenntnisse zur Ausprägung und Verlauf der Kc-Werte unter Freilandbedingungen, sowie deren Messung mit Hilfe der Eddy-Kovarianz. Aufbauend auf diesen Versuchen können weitere kritische Kulturen überprüft werden und

das grundsätzliche Vertrauen auf die Plausibilität der Geisenheimer Kc-Werte gestärkt werden.

Somit steht nun mit GSEHEN der gemüsebaulichen Bewässerungsplanung ein geeignetes und zuverlässiges Tool zur Verfügung, in Hinsicht auf gegenwärtige und zukünftige Herausforderungen.

Zum Zeitpunkt der Erstellung des Abschlussberichts mussten von der Firma CGS noch Korrekturen an der Software GSEHEN vorgenommen werden. Sobald diese erfolgt sind, wird die Software auf dieser Internetseite veröffentlicht:

https://www.hs-geisenheim.de/forschung/institute/gemuesebau/ueberblick-institut-fuer-gemuesebau/bewaesserung/geisenheimer-bewaesserungssteuerung/

6 Literaturverzeichnis

Allen R. G. et al. (1998): Crop Evapotranspiration-Guidelines for Computing Crop Water. Requirements-FAO Irrigation and Drainage Paper 56. In: FAO, Rome 300, S. 6541.

Bryla D.R., Trout J., Ayars J.E., (2010): Weighing Lysimeters for Developing Crop Coefficients and Efficient Irrigation Practices for Vegetable Crops. HortScience 45, 1597–1604. https://doi.org/10.21273/HORTSCI.45.11.1597

Cahn M.D., Johnson L.F. (2017). New Approaches to Irrigation Scheduling of Vegetables. Horticulturae 3. https://doi.org/10.3390/horticulturae3020028

Elia, A., Conversa, G. (2015): A decision support system (GesCoN) for managing fertigation in open field vegetable crops. Part I-methodological approach and description of the software. Front. Plant Sci. 6. https://doi.org/10.3389/fpls.2015.00319

Hartmann H.D., Pfülb E., Zengerle K.H. (2000): Wasserverbrauch uns Bewässerung von Gemüse, Geisenheimer Berichte der Forschungsanstalt Geisenheim. Geisenheim.

Mirás-Avalos J., Rubio-Asensio J., Ramírez-Cuesta J., Maestre-Valero J., Intrigliolo D. (2019): Irrigation-Advisor - A Decision Support System for Irrigation of Vegetable Crops. Water 11, 2245. https://doi.org/doi:10.3390/w11112245

Moneo, M., Iglesias, A. (2007): A framework for Irrigation management during drought: Application in two case studies in the Tagus Basin, Spain. Options Méditerranéennes : Série B. Etudes et Recherches 56, 305–320.

Olberz M., Zinkernagel J. (2014): Die Geisenheimer Bewässerungssteuerung - heute und morgen. ZVG-Gartenbau-Report 40 (7/8) S. 23 - 25.

Pardossi A., Incrocci L. (2011): Traditional and New Approaches to Irrigation Scheduling in Vegetable Crops. Horttechnology 21, 309–313. https://doi.org/10.21273/HORTTECH.21.3.309.

Paschold P.J., Kleber J., Mayer N. (2002): Geisenheimer Bewässerungssteuerung. Zeitschrift für Bewässerungswirtschaft 37 (1) S. 5 - 15.

Paschold P.J., Frühauf C., Schaller J., Kleber J., Mayer N. (2011): Geisenheimer Bewässerungssteuerung und FAO-Grasverdunstung. Deutsche Gartenbauwissenschaftliche Gesellschaft S. 1 - 5. DOI: 10.5288/dgg-pr-01-05-nm-2011

R Core Team (2017). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Smith, M. (1992): CROPWAT: A Computer Program for Irrigation Planning and Management, Ver. 5.7. ed, FAO Irrigation and Drainage Paper. Food and Agriculture Organization of the United Nations, Rome, Italy.

Tsirogiannis I.L., Malamos N., Christofides A., Anastasiadis S., Koliopanos C., Fotia K., Baltzoi, P. (2018): Pilot operation and evaluation of a meteorological data fed water budget system for turfgrass. Acta Hort. 1197, 195–202. https://doi.org/10.17660/ActaHortic.2018.1197.26

Zinkernagel J., Kaim E., Artelt B., Reiche S. (2012): Bewässerung bei Spargel wichtig. Gemüse 48, 8, 20–21.

Zinkernagel et al., (2019): Geisenheimer Bewässerungssteuerung. https://www.hs-geisenheim.de/fileadmin/redaktion/FORSCHUNG/Institut_fuer_Gemuesebau/Ueberblick_Institut_fuer_Gemuesebau/Geisenheimer_Steuerung/kc-Werte_FAO_Grasverdunstung_2019.pdf

7 Anhang

7.1 Ausschreibung und Leistungsverzeichnis

Specifications

Project GSEHEN

Geisenheim University | Department for Vegetable Crops

April 7, 2017

1 Introduction

The Geisenheim University (HGU) Department for Vegetable Crops tenders the development of a software application. The following specifications describe the design constraints of an irrigation scheduling decision support software implementing the Geisenheim Irrigation Scheduling [1]. Its development and programming is intended to be accomplished in collaboration between Geisenheim University and an external software house. The finished software is supposed to be released under an open source compatible licence.

2 Problem Statement

The economic success in horticulture is directly linked to irrigation. More than in any other agricultural field, irrigation determines product quality and quantity. Further, in contrast to other agricultural disciplines, a vegetable grower produces several crops in relatively short reoccurring cycles on different fields or field plots, simultaneously. Irrigation of those crops has to be planned, documented and managed on a daily basis. At the same time, the irrigation management coincides with other cultivation practices, which vary similarly with the cultivation needs. Consequently, the grower has little time for the irrigation management process. Therefore, he is in need for an accessible tool which visualizes all the required information for his irrigation scheduling decisions fast and accurately. In general, irrigation, if judged by the rule of thumb, is managed with a predominant overestimation of irrigation needs. This leads to water waste and nutrient leakage.
To overcome these practices and to provide a solution which is accepted by the majority of farmers, the irrigation scheduling tool needs a scientifically and practically proven foundation. Further,

[1] http://www.hs-geisenheim.de/forschungszentren/institut-fuer-gemuesebau/forschung/geisenheimer-steuerung.html

cultivation systems and farm profiles diverse in major aspects, consequently the irrigation management tool needs to be as modular and customizable as possible. Critical aspects are e.g. the provision of weather data as well as farm- or site-specific correction factor for irrigation. GS, a decision support system (DSS) for irrigation management, fulfils these requirements. However, GS is not yet implemented in an accessible way to contribute to irrigation management tools. Thus, we see the need to implement GS in an open, well documented and for irrigation management tools viable way. Another important factor in this context is our duty to contribute our knowledge, in particular the GS irrigation scheduling management for vegetable crops to a widespread audience. Consequently, all software developments will be published under open source licences. This implies a conformity to these licences for all programming environments, packages or compilers used to program the software. Furthermore, we aspire to publish the software implementation with a comprehensive documentation as an example for further implementations of GS in existing farm management tools to spread the knowledge of GS beyond this projects reach.
Summarizing, we see the need to provide vegetable growers with a tool to quickly and easily manage and decide daily irrigation schedules of multiple fields and crops based on GS in a flexible open source software. Further, we strive to raise GS on a modern, appropriate documented and accessible basis, to allow a further distribution into other software systems and with this the practical application.

3 Specification of the software

3.1 Functionality. What is the software supposed to do?

The software is an irrigation scheduling DSS with integrated field management. Its internal calculations are based on an implementation of the GS algorithms (see figure 1). The software allows the user to manage and visualize the irrigation needs of multiple fields with their underlying field plots. It allows for management of several field plots on a field with changing dimensions and cultivated crops over the year. To calculate GS and supply decision support numerous data inputs are necessary and need to be stored locally (e.g. sqlite) for the ongoing process of irrigation recommendation calculation and the user fed irrigation management. The data the software needs to process can be divided into:

- base data (crop, soil)
- user supplied data (field, field plots, location, shape, crop, soil type and profile, weather data, irrigation events)
- automatically generated and acquired data (GS calculation algorithms, weather data)

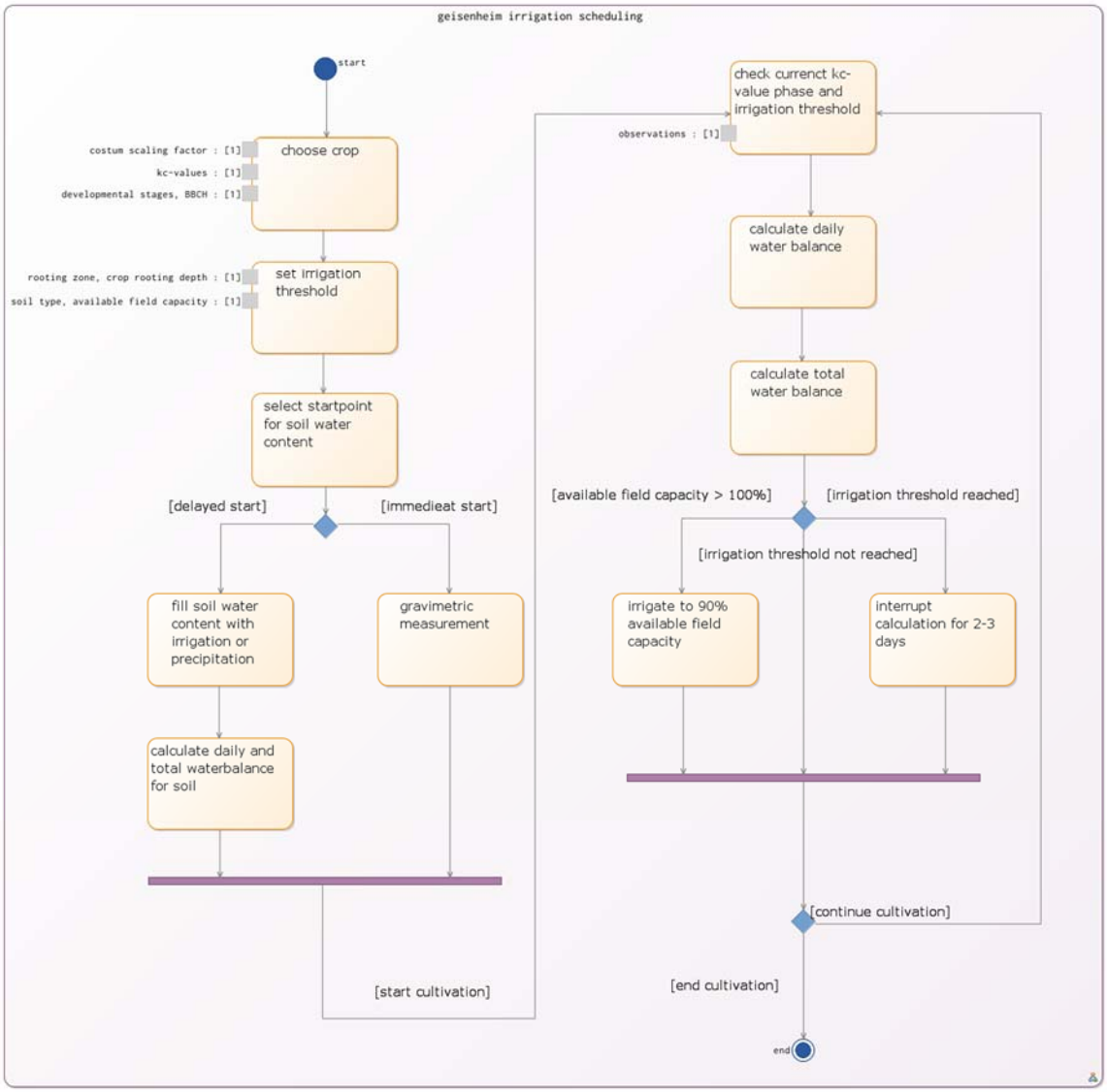

Figure 1: Geisenheim Irrigation Scheduling algorithm

A critical source of information is the weather data. It can be provided from a large pool of sources, so the management tool should be capable of dealing with the most common data formats including an automatic import from weather stations. So, there is an interface to get the needed weather data automatically and directly from weather stations on a daily basis, with following calculation of need climatic parameters. A major climatic parameter is the daily FAO56 reference evapotranspiration as provided by Allen et al. 1998, which must be calculated from weather data. Additionally, there is a system to import weather data for text files (e.g., .csv/.xml). Furthermore, the implementation of weather data sources should be modular plugin based to be future proof. Since weather data is a local measurement, weather data sources should be adjustable per field or organisation unit. Further, it must be possible to introduce a firm-specific correction factor globally, per field or organisation unit. A main feature of the software is the "always up to date" base data source, which is acquired through updating the base data from an external database server on a regular basis. The server is a PostgreSQL database maintained and updated by the HGU and therefore it represents the up-to-date base data sources. The base data includes all the information about available crops and their parameters needed for GS calculation. Furthermore, it stores information about the properties of different soil types. The specifications from the soil and crop data are displayed in figure 3. Database updates are associated with the control and regulation of for the user available crop or soil factors. Individual inputs like an irrigation pause after a heavy rain event for a defined amount of time e.g. two days should also find consideration by the management tool. For field or field plot configuration it is necessary to define the field's soil profile. The soil profile is typically defined by three horizons (e.g 0-30 cm, 30-60 cm and 60-90 cm) with their respective soil types, but should be modifiable individually (see figure 3: Soil, Soil_Profile, Soil_Profile_Depth). The software is accessible via an extensive, feature-rich user interface, which allows for adjustment and manipulation of every aspect of the irrigation management process described. Furthermore, the GUI provides a summarizing overview in a graphical representation of the fields/management unit/farm, including geospatial information and representations on the farm level. The major use cases, which need to be covered by the software are displayed in figure 2. Furthermore, figure 3 provides a brief overview of possible classes and attributes, which may be needed for the irrigation scheduling software. **All figures are examples for a better overview and clarification of the problem but are in no way intended to represent a comprehensive list of all elements needed for the software. Final design patterns are intended to be accomplished in collaboration between Geisenheim University and the software house.**

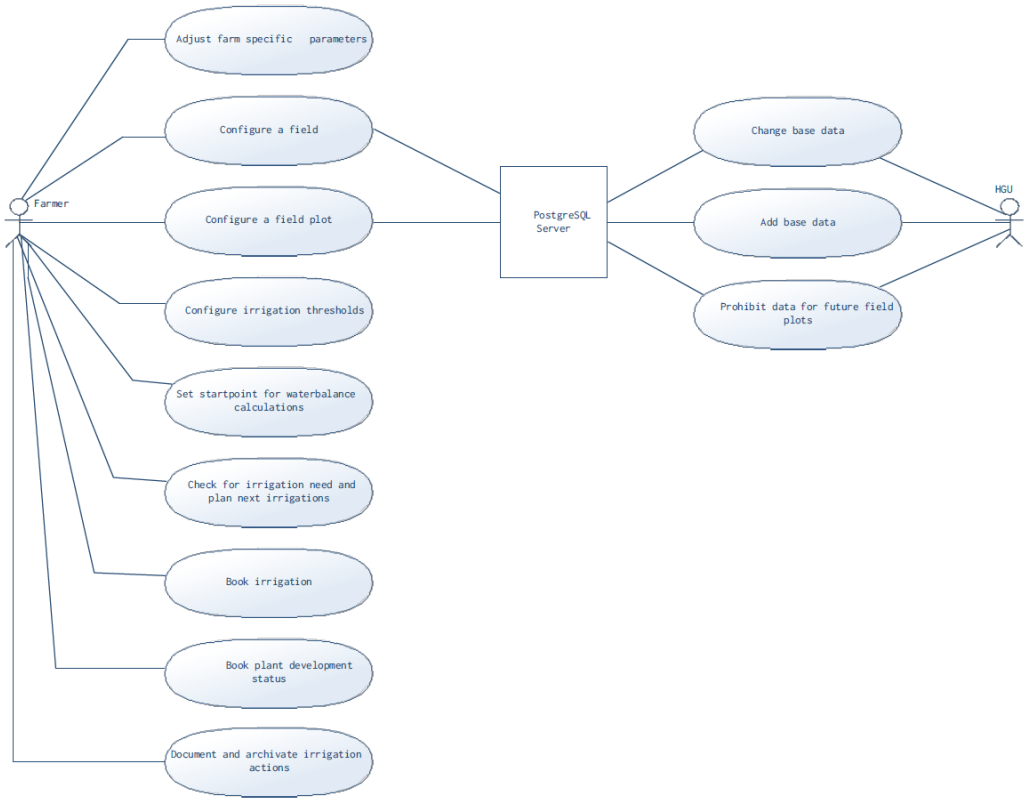

Figure 2: Use case diagram for the Software displaying the major use cases, which need to be covered by the software. Actors are the farmer, the PostgreSQL server, and Geisenheim University (HGU).

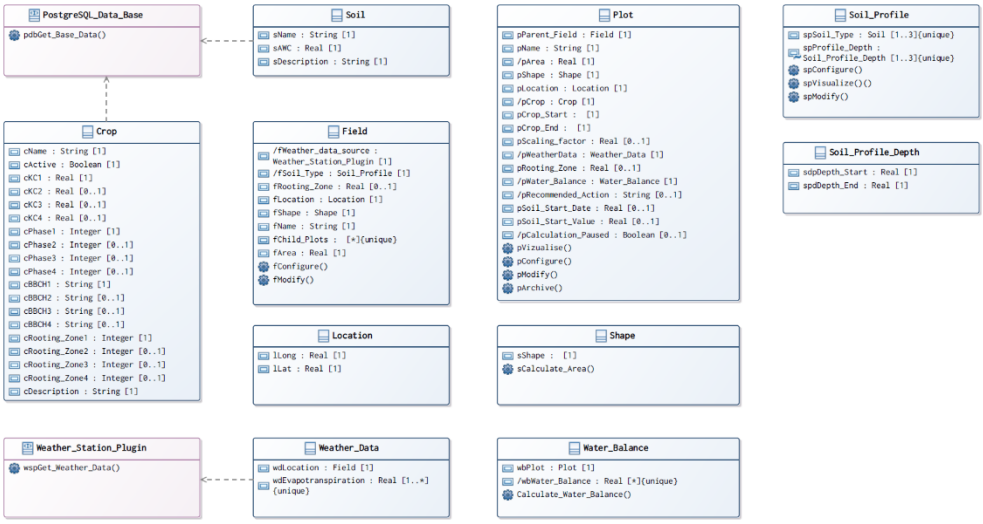

Figure 3: Class diagram with possible classes and attributes needed for the irrigation scheduling software.

3.2 External interfaces. How does the software interact with people, the system's hardware, other hardware, and other software?

- There is a mandatory connection with the kc and soil database of the HGU.
- There are external interfaces for weather data as part of a plugin system.
- It has an interactive feature rich graphical user interface (GUI) which display irrigation recommendations as geospatial informations.

3.3 Performance. What is the speed, availability, response time, recovery time of various software functions, etc.?

Weather data updates and irrigation recommendations should be available in timely manner, so that the user experience is not influenced negatively. All information necessary for irrigation scheduling, should be available on program start up.

3.4 Attributes. What are the portability, correctness, maintainability, security considerations?

Security The software handles valuable firm specific informations. So, the locally stored data should be encrypted, and the access to the software securely password protected.

Correctness All calculations should be maintained under highest possible precision (e.g. double precision). Further, all inputs by the software use should be checked (fail safe). Furthermore, the used weather data should be checked for sanity. Bad or wrong weather data has a massive impact on the correctness of the outcomes of the DSS. Additionally, with implementation of the regular update of base data, user options for the selection of base data are only available on the permissions granted by the base data server maintained by the HGU (forms).

3.5 Design constraints imposed on an implementation. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

The finished software is supposed to be released under an open source compatible licence. Target Systems are home computers with, Windows, Linux and MacOS. It is mandatory to build a comprehensive documentation of all software parts, especially of the developed plugin system. Further, for licensing purposes, there is a list with used packages/libraries and their licences included.

7.2 Grunddaten von GSEHEN

Tabelle 2: Grunddaten von GSEHEN (Auszug; cKC1-4: Kc-Wert in der Entwicklungsphase 1-4; cPhase1-4: Dauer der Entwicklungsphase in Tagen (100 = fehlende Daten); cRooting_Zone1-4: Bodentiefe, die in den Entwicklungsstadien 1-4 betrachtet werden)

basename	cKC1	cKC2	cKC3	cKC4	cPhase1	cPhase2	cPhase3	cPhase4	cRooting_Zone1	cRooting_Zone2	cRooting_Zone3	cRooting_Zone4
asparagus	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
beetroot	0,5	1,1	1,6	1,8	100	100	100	100	30	60	60	60
broadbeans	0,7	1,1	1,6	1,8	100	100	100	100	30	60	60	60
broccoli	0,7	1,1	1,8	NA	22	31	38	NA	30	60	60	NA
brusselssprout	0,7	1,1	1,6	1,8	100	100	100	100	30	60	60	60
carrot	0,4	0,8	1,1	NA	46	24	60	NA	30	60	90	NA
cauliflower	0,7	1,1	1,6	NA	27	24	34	NA	30	60	60	NA
celery	0,7	1,1	1,5	1,8	28	20	30	69	30	60	60	60
chinesecabbage	0,7	1,1	1,6	NA	16	19	42	NA	30	60	60	NA
corn	0,5	0,6	0,75	1	41	25	26	102	30	60	60	90
dwarfbean	0,4	1,1	1,3	NA	100	100	100	NA	30	60	60	NA
endive	0,7	1,1	1,6	NA	100	100	100	NA	30	30	30	NA
fennel	0,7	1,3	1,8	NA	24	19	58	NA	30	60	60	NA
gherkins	0,7	1,1	1,5	NA	100	100	100	NA	30	30	30	NA
headcabbage	0,7	0,8	1	1,1	30	10	27	96	30	60	60	60
iceberglettuce	0,7	1,1	1,3	1,6	19	32	12	59	30	30	30	30
kale	0,7	1,1	1,6	1,8	11	15	18	100	30	60	60	60
leek	0,7	1,1	1,6	1,8	63	17	10	49	30	60	60	60
lettuce	0,7	1,1	1,6	NA	11	40	40	30	30	30	30	60
onion	0,7	1,3	1,6	0	10	50	30	100	30	60	60	NA
parsley	0,4	1,1	1,3	NA	66	56	77	NA	30	60	60	NA
potato	0,5	0,8	1,1	0	39	24	43	39	30	60	60	60
radish	0,7	1,1	1,3	NA	18	22	10	NA	15	15	15	NA
sugarbeet	0,3	0,5	0,8	1,1	100	100	100	100	30	60	60	90
tomato	0,7	1,1	1,6	NA	100	100	100	NA	30	60	60	NA
wintercrop	0,5	0,9	0	NA	100	100	100	NA	30	60	90	NA
zucchini	0,7	1,1	1,6	NA	35	21	49	NA	30	60	60	NA

7.3 Übersetzungen

Tabelle 3: Namen der Kulturparameter in Englisch (en) und Deutsch (de)

KEY	LOCALE_ID	TEXT
onion.name	de	Zwiebel
onion.name	en	Onion
onion.bbch1	de	Keimung
onion.bbch1	en	Germination
onion.bbch2	de	4 Blätter
onion.bbch2	en	4 Leaves
onion.bbch3	de	6 Blätter
onion.bbch3	en	6 Leaves
onion.description	de	Sommerzwiebel
onion.description	en	Summer onion
lettuce.name	de	Salat
lettuce.name	en	Lettuce
lettuce.bbch1	de	Pflanzung
lettuce.bbch1	en	Planting
lettuce.bbch2	de	30% Bedeckung
lettuce.bbch2	en	30% Crop cover
lettuce.bbch3	de	60% Bedeckung
lettuce.bbch3	en	60% Crop cover
lettuce.bbch4	de	80% Bedeckung
lettuce.bbch4	en	80% Crop cover
lettuce.description	de	Industriesalat Kopfgewicht >500g
lettuce.description	en	Industrial lettuce, head weight > 500 g
cauliflower.name	de	Blumenkohl
cauliflower.name	en	Cauliflower
cauliflower.bbch1	de	ab Pflanzung
cauliflower.bbch1	en	Planting
cauliflower.bbch2	de	Ab 8. Blatt
cauliflower.bbch2	en	8. Leaf
cauliflower.bbch3	de	Ab 70% des Pflanzendurchmessers
cauliflower.bbch3	en	70% Plant diameter
cauliflower.description	de	NA
cauliflower.description	en	NA

KEY	LOCALE_ID	TEXT
broccoli.name	de	Brokkoli
broccoli.name	en	Broccoli
broccoli.bbch1	de	ab Pflanzung
broccoli.bbch1	en	Planting
broccoli.bbch2	de	Ab 8. Blatt
broccoli.bbch2	en	8. Leaf
broccoli.bbch3	de	Ab 14. Blatt
broccoli.bbch3	en	14. Leaf
broccoli.description	de	NA
broccoli.description	en	NA
dwarfbean.name	de	Buschbohnen
dwarfbean.name	en	Dwarfbean
dwarfbean.bbch1	de	ab Auflaufen
dwarfbean.bbch1	en	Germination
dwarfbean.bbch2	de	ab Blühbeginn
dwarfbean.bbch2	en	Floration
dwarfbean.bbch3	de	1. Hülse hat volle Länge
dwarfbean.bbch3	en	1. Pulse has final length
dwarfbean.description	de	NA
dwarfbean.description	en	NA
chinesecabbage.name	de	Chinakohl
chinesecabbage.name	en	Chinese cabbage
chinesecabbage.bbch1	de	ab Pflanzung
chinesecabbage.bbch1	en	Planting
chinesecabbage.bbch2	de	ab 6. Blatt
chinesecabbage.bbch2	en	6. Leaf
chinesecabbage.bbch3	de	Kopfbildung beginnt
chinesecabbage.bbch3	en	Being of head formation
chinesecabbage.description	de	NA
chinesecabbage.description	en	NA
iceberglettuce.name	de	Eissalat
iceberglettuce.name	en	Iceberg lettuhce
iceberglettuce.bbch1	de	ab Pflanzung
iceberglettuce.bbch1	en	Planting
iceberglettuce.bbch2	de	ab 7. bis 9. Blatt
iceberglettuce.bbch2	en	7. to 9. Leaf
iceberglettuce.bbch3	de	Kopfbildung beginnt
iceberglettuce.bbch3	en	Being of head formation
iceberglettuce.bbch4	de	30% der Kopfgröße erreicht
iceberglettuce.bbch4	en	30% Head diameter
iceberglettuce.description	de	NA
iceberglettuce.description	en	NA
endive.name	de	Endivien
endive.name	en	Endive
endive.bbch1	de	ab Pflanzung

KEY	LOCALE_ID	TEXT
endive.bbch1	en	Planting
endive.bbch2	de	ab 7. Blatt
endive.bbch2	en	7. Leaf
endive.bbch3	de	Ab 10. Blatt
endive.bbch3	en	10. Leaf
endive.description	de	NA
endive.description	en	NA
fennel.name	de	Fenchel, Knollen-
fennel.name	en	Fennel, tuber
fennel.bbch1	de	ab Pflanzung
fennel.bbch1	en	Planting
fennel.bbch2	de	ab 7. Blatt
fennel.bbch2	en	7. Leaf
fennel.bbch3	de	ab 8. Blatt
fennel.bbch3	en	8. Leaf
fennel.description	de	NA
fennel.description	en	NA
wintercrop.name	de	Getreide, Winter
wintercrop.name	en	Crop, winter
wintercrop.bbch1	de	ab Auflaufen
wintercrop.bbch1	en	Germination
wintercrop.bbch2	de	ab Schossen
wintercrop.bbch2	en	Shooting
wintercrop.bbch3	de	ab beginnender Teigreife
wintercrop.bbch3	en	Begin of dough stage
wintercrop.description	de	NA
wintercrop.description	en	NA
kale.name	de	Grünkohl
kale.name	en	Kale
kale.bbch1	de	ab Pflanzung
kale.bbch1	en	Planting
kale.bbch2	de	ab 6. Laubblatt
kale.bbch2	en	6. Leaf
kale.bbch3	de	ab verstärkter Blattbildung
kale.bbch3	en	Increased leaf formation
kale.bbch4	de	ab Bestandesschluss
kale.bbch4	en	Crop establishment
kale.description	de	NA
kale.description	en	NA
gherkins.name	de	Gurken, Einlege-
gherkins.name	en	Gherkins
gherkins.bbch1	de	ab Auflaufen
gherkins.bbch1	en	Germination
gherkins.bbch2	de	ab Blühbeginn
gherkins.bbch2	en	Floration

KEY	LOCALE_ID	TEXT
gherkins.bbch3	de	ab Erntebeginn
gherkins.bbch3	en	Begin of harvest
gherkins.description	de	NA
gherkins.description	en	NA
potato.name	de	Kartoffel
potato.name	en	Potato
potato.bbch1	de	Sprosse durchbrechen den Boden
potato.bbch1	en	Shoots break trough the soil
potato.bbch2	de	ab 2. Trieb > 5 cm lang
potato.bbch2	en	2. Shoot < 5 cm
potato.bbch3	de	ab Bestandesschluss
potato.bbch3	en	Crop establishment
potato.bbch4	de	ab Blattvergilbung
potato.bbch4	en	Leaf yellowing
potato.description	de	NA
potato.description	en	NA
headcabbage.name	de	Kopfsalat
headcabbage.name	en	Head cabbage
headcabbage.bbch1	de	ab Pflanzung
headcabbage.bbch1	en	Planting
headcabbage.bbch2	de	ab 8. Blatt
headcabbage.bbch2	en	8. Leaf
headcabbage.bbch3	de	ab 11. Blatt
headcabbage.bbch3	en	11. Leaf
headcabbage.bbch4	de	ab beginnender Kopfbildung
headcabbage.bbch4	en	Begin of head formation
headcabbage.description	de	NA
headcabbage.description	en	NA
corn.name	de	Mais, Körner- ,Zucker-
corn.name	en	Corn
corn.bbch1	de	ab Auflaufen
corn.bbch1	en	Germination
corn.bbch2	de	ab Höhe 0,50 m
corn.bbch2	en	Heigth 0,50 m
corn.bbch3	de	ab Höhe 1,00 m
corn.bbch3	en	Height 1,00 m
corn.bbch4	de	ab Höhe 1,50 m
corn.bbch4	en	Height 1,50 m
corn.description	de	NA
corn.description	en	NA
carrot.name	de	Möhren
carrot.name	en	Carrot
carrot.bbch1	de	ab Auflaufen
carrot.bbch1	en	Germination
carrot.bbch2	de	ab 5. Blatt

KEY	LOCALE_ID	TEXT
carrot.bbch2	en	5. Leaf
carrot.bbch3	de	ab Bestandesschluss
carrot.bbch3	en	Crop establishment
carrot.description	de	NA
carrot.description	en	NA
parsley.name	de	Petersielie
parsley.name	en	Parsley
parsley.bbch1	de	ab Auflaufen
parsley.bbch1	en	Germination
parsley.bbch2	de	ab 5. Blatt
parsley.bbch2	en	5. Leaf
parsley.bbch3	de	nach der 1. Ernte
parsley.bbch3	en	After 1. harvest
parsley.description	de	NA
parsley.description	en	NA
leek.name	de	Porree
leek.name	en	Leek
leek.bbch1	de	ab Pflanzung
leek.bbch1	en	Planting
leek.bbch2	de	ab Schaftdurchmesser 13 mm
leek.bbch2	en	Shaft diameter 13 mm
leek.bbch3	de	ab Schaftdurchmesser 16 mm
leek.bbch3	en	Shaft diameter 16 mm
leek.bbch4	de	ab Schaftdurchmesser 20 mm
leek.bbch4	en	Shaft diameter 20 mm
leek.description	de	NA
leek.description	en	NA
broadbeans.name	de	Puffbohnen
broadbeans.name	en	Broad beans
broadbeans.bbch1	de	ab Auflaufen
broadbeans.bbch1	en	Germination
broadbeans.bbch2	de	ab Höhe 10 cm
broadbeans.bbch2	en	Height 10 cm
broadbeans.bbch3	de	ab Blühbeginn
broadbeans.bbch3	en	Floration
broadbeans.bbch4	de	ab Hülsenansatz
broadbeans.bbch4	en	Pusle initiation
broadbeans.description	de	NA
broadbeans.description	en	NA
radish.name	de	Rettich
radish.name	en	Radish
radish.bbch1	de	ab Auflaufen
radish.bbch1	en	Germination
radish.bbch2	de	ab Beginn Dickenwachstum
radish.bbch2	en	Begin of width growth

KEY	LOCALE_ID	TEXT
radish.bbch3	de	ab 50% Rübendurchmesser
radish.bbch3	en	50% Beet diameter
radish.description	de	NA
radish.description	en	NA
brusselssprout.name	de	Rosenkohl
brusselssprout.name	en	Brusselssprout
brusselssprout.bbch1	de	ab Pflanzung
brusselssprout.bbch1	en	Planting
brusselssprout.bbch2	de	ab 6. Blatt
brusselssprout.bbch2	en	6. Leaf
brusselssprout.bbch3	de	ab Bestandesschluss
brusselssprout.bbch3	en	Crop establishment
brusselssprout.bbch4	de	ab Röschenansatz
brusselssprout.bbch4	en	Florent initiation
brusselssprout.description	de	NA
brusselssprout.description	en	NA
beetroot.name	de	Rote Bete
beetroot.name	en	Beet root
beetroot.bbch1	de	ab Auflaufen
beetroot.bbch1	en	Germination
beetroot.bbch2	de	ab 4. Blatt
beetroot.bbch2	en	4. Leaf
beetroot.bbch3	de	ab 8. Blatt
beetroot.bbch3	en	8. Leaf
beetroot.bbch4	de	ab Bestandesschluss
beetroot.bbch4	en	Crop establishment
beetroot.description	de	NA
beetroot.description	en	NA
celery.name	de	Sellerie, -Knollen
celery.name	en	Celery
celery.bbch1	de	ab Pflanzung
celery.bbch1	en	Planting
celery.bbch2	de	ab 7. Blatt
celery.bbch2	en	7. Leaf
celery.bbch3	de	ab beginnender Knollenentwicklung
celery.bbch3	en	Begin of tuber formation
celery.bbch4	de	ab Bestandesschluss
celery.bbch4	en	Crop establishment
celery.description	de	NA
celery.description	en	NA
asparagus.name	de	Spargel
asparagus.name	en	Asparagus
asparagus.description	de	NA
asparagus.description	en	NA
tomato.name	de	Tomate, Freiland

KEY	LOCALE_ID	TEXT
tomato.name	en	Tomato, open field
tomato.bbch1	de	ab Pflanzung
tomato.bbch1	en	Planting
tomato.bbch2	de	ab Höhe 0,75 m
tomato.bbch2	en	Height 0,75 m
tomato.bbch3	de	ab Höhe 1,00 m
tomato.bbch3	en	Height 1,00 m
tomato.description	de	NA
tomato.description	en	NA
zucchini.name	de	Zucchini
zucchini.name	en	Zucchini
zucchini.bbch1	de	ab Auflaufen
zucchini.bbch1	en	Germination
zucchini.bbch2	de	ab Blühbeginn
zucchini.bbch2	en	Floration
zucchini.bbch3	de	ab Bestandesschluss
zucchini.bbch3	en	Crop establishment
zucchini.description	de	NA
zucchini.description	en	NA
sugarbeet.name	de	Zuckerrübe
sugarbeet.name	en	Sugar beet
sugarbeet.bbch1	de	ab Auflaufen
sugarbeet.bbch1	en	Germination
sugarbeet.bbch2	de	ab 5. Blatt
sugarbeet.bbch2	en	5. Leaf
sugarbeet.bbch3	de	ab Bestandesschluss
sugarbeet.bbch3	en	Crop establishment
sugarbeet.bbch4	de	ab Rübendurchmesser 12 cm bis E IX
sugarbeet.bbch4	en	Beet diameter 12 cm until end IX
sugarbeet.description	de	NA
sugarbeet.description	en	NA

7.4 Quellcode

B.1 AbsTemp.java

```
package de.hgu.gsehen.evapotranspiration;

public class AbsTemp {
  /**
   * This class combines all absolute Temperatures.
   *
   * @param absTempMean absolute Temperature mean
   * @param absTempMax absolute temperature maximum
   * @param absTempMin absolute temperature minimum
   */
  public AbsTemp(double absTempMean, double absTempMax, double absTempMin) {
    super();
    this.absTempMean = absTempMean;
    this.absTempMax = absTempMax;
    this.absTempMin = absTempMin;
  }

  private double absTempMean;
  private double absTempMax;
  private double absTempMin;

  public double getAbsTempMean() {
    return absTempMean;
  }

  public void setAbsTempMean(double absTempMean) {
    this.absTempMean = absTempMean;
  }

  public double getAbsTempMax() {
    return absTempMax;
  }

  public void setAbsTempMax(double absTempMax) {
    this.absTempMax = absTempMax;
  }

  public double getAbsTempMin() {
    return absTempMin;
  }

  public void setAbsTempMin(double absTempMin) {
    this.absTempMin = absTempMin;
  }


}
```

B.2 DayData.java

```
package de.hgu.gsehen.evapotranspiration;
```

```java
import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

/**
 * Class to represent the environmental data to calculate the GS water balance.
 *
 * @author MO
 *
 */
@Entity
public class DayData implements Comparable<DayData> {

  @Id
  @GeneratedValue
  private long id;
  private Date date;
  private double tempMean;
  private Double tempMin;
  private Double tempMax;
  private double airHumidityRelMean;
  private Double airHumidityRelMin;
  private Double airHumidityRelMax;
  private double globalRad;
  private Double precipitation;
  private double windspeed2m;
  private Double et0;
  private Double currentKc;
  private Double etc;
  private Double irrigation;
  private Double dailyBalance;
  private Integer currentRootingZone;
  private Double currentAvailableSoilWater;
  private Double currentTotalWaterBalance;

  public Double getCurrentKc() {
    return currentKc;
  }

  public void setCurrentKc(Double currentKc) {
    this.currentKc = currentKc;
  }


  public Double getEtc() {
    return etc;
  }

  public void setEtc(Double etc) {
    this.etc = etc;
  }


  public Double getIrrigation() {
```

```
    return irrigation;
}

public void setIrrigation(Double irrigation) {
    this.irrigation = irrigation;
}


public Double getDailyBalance() {
    return dailyBalance;
}

public void setDailyBalance(Double dailyBalance) {
    this.dailyBalance = dailyBalance;
}

public Date getDate() {
    return date;
}

public void setDate(Date date) {
    this.date = date;
}

public double getTempMean() {
    return tempMean;
}

public void setTempMean(double tempMean) {
    this.tempMean = tempMean;
}

public Double getTempMin() {
    return tempMin;
}

public void setTempMin(Double tempMin) {
    this.tempMin = tempMin;
}

public Double getTempMax() {
    return tempMax;
}

public void setTempMax(Double tempMax) {
    this.tempMax = tempMax;
}

public double getAirHumidityRelMean() {
    return airHumidityRelMean;
}

public void setAirHumidityRelMean(Double airHumidityRelMean) {
    this.airHumidityRelMean = airHumidityRelMean;
}
```

```
public Double getAirHumidityRelMin() {
  return airHumidityRelMin;
}

public void setAirHumidityRelMin(Double airHumidityRelMin) {
  this.airHumidityRelMin = airHumidityRelMin;
}

public Double getAirHumidityRelMax() {
  return airHumidityRelMax;
}

public void setAirHumidityRelMax(Double airHumidityRelMax) {
  this.airHumidityRelMax = airHumidityRelMax;
}

public double getGlobalRad() {
  return globalRad;
}

public void setGlobalRad(double globalRad) {
  this.globalRad = globalRad;
}

public Double getPrecipitation() {
  return precipitation;
}

public void setPrecipitation(Double precipitation) {
  this.precipitation = precipitation;
}

public double getWindspeed2m() {
  return windspeed2m;
}

public void setWindspeed2m(double windspeed2m) {
  this.windspeed2m = windspeed2m;
}

public Double getEt0() {
  return et0;
}

public void setEt0(Double et0) {
  this.et0 = et0;
}

public Integer getCurrentRootingZone() {
  return currentRootingZone;
}

public void setCurrentRootingZone(Integer currentRootingZone) {
  this.currentRootingZone = currentRootingZone;
}
```

```
public Double getCurrentAvailableSoilWater() {
  return currentAvailableSoilWater;
}

public void setCurrentAvailableSoilWater(Double currentAvailableSoilWater) {
  this.currentAvailableSoilWater = currentAvailableSoilWater;
}

public Double getCurrentTotalWaterBalance() {
  return currentTotalWaterBalance;
}

public void setCurrentTotalWaterBalance(Double currentTotalWaterBalance) {
  this.currentTotalWaterBalance = currentTotalWaterBalance;
}

@Override
public int compareTo(DayData o) {
  return getDate().compareTo(o.getDate());
}

/**
 * Constructor DayData class.
 *
 * @author MD
 *
 * @param date is the actual date
 * @param tempMean Temperature mean in °C
 * @param tempMin Temperature minimum in °C
 * @param tempMax Temperature maximum in °C
 * @param airHumidityRelMean Relative air humidity in percent mean
 * @param airHumidityRelMin Relative air humidity in percent min
 * @param airHumidityRelMax Relative air humidity in percent max
 * @param globalRad Global radiation in MJ
 * @param precipitation Rainfall
 * @param windspeed2m Mean windspeed at 2 meters above ground level in m/s
 * @param et0 daily evapotranspiration in mm
 * @param currentKc is the days kc value
 * @param dailyBalance is the daily Water blance in mm
 * @param currentRootingZone is the current rooting zone of the subordinated crop
 * @param currentAvailableSoilWater is the available soil water at this state
 * @param currentTotalWaterBalance is the current totla water blance of the crop
 */
public DayData(Date date, double tempMean, Double tempMin, Double tempMax,
    double airHumidityRelMean, Double airHumidityRelMin, Double airHumidityRelMax,
    double globalRad, Double precipitation, double windspeed2m, Double et0, Double
        currentKc,
    Double etc, Double irrigation, Double dailyBalance, Integer currentRootingZone,
    Double currentAvailableSoilWater, Double currentTotalWaterBalance) {
  super();
  this.date = date;
  this.tempMean = tempMean;
  this.tempMin = tempMin;
  this.tempMax = tempMax;
  this.airHumidityRelMean = airHumidityRelMean;
  this.airHumidityRelMin = airHumidityRelMin;
```

```
    this.airHumidityRelMax = airHumidityRelMax;
    this.globalRad = globalRad;
    this.precipitation = precipitation;
    this.windspeed2m = windspeed2m;
    this.et0 = et0;
    this.currentKc = currentKc;
    this.etc = etc;
    this.irrigation = irrigation;
    this.dailyBalance = dailyBalance;
    this.currentRootingZone = currentRootingZone;
    this.currentAvailableSoilWater = currentAvailableSoilWater;
    this.currentTotalWaterBalance = currentTotalWaterBalance;
  }

  public DayData() {}


}
```

B.3 EnvCalculator.java

```
package de.hgu.gsehen.evapotranspiration;

import static java.lang.Math.PI;
import static java.lang.Math.acos;
import static java.lang.Math.cos;
import static java.lang.Math.exp;
import static java.lang.Math.pow;
import static java.lang.Math.sin;
import static java.lang.Math.sqrt;
import static java.lang.Math.tan;

import java.util.Calendar;

import org.junit.platform.commons.annotation.Testable;

public class EnvCalculator {
  private static final double ALPHA = 0.23;
  private static final double GSC = 0.082;
  /**
   * Flag for calculating Rs0.
   */
  private static final boolean OLDRS0 = false;
  /**
   * Soil heat flux is 0 for daily calculation step.
   */
  private static final double G = 0;
  /**
   * Flag for calculating gamma procedure.
   */
  private static final boolean OLDGAMMA = false;
  /**
   * Flag for calculating net longwave radiation.
   */
  private static final boolean OLDRNL = false;

  /**
```

```
 * Main method to calculate the daily reference evapotranspiration as published by
     Allen et al.
 * All parameters are daily values. method changes object DayData et0
 *
 * @author MD
 * @param dayData An Object of the DayData class
 * @param geoData An Object of the GeoData class
 */
@Testable
public static void calculateEt0(DayData dayData, GeoData geoData) {
  Integer yday = tellDoy(dayData);

  double psi = convertDegToRad(geoData);
  double solarRad = dayData.getGlobalRad();
  double latentVaporHeatFlux = calculateLatentVaporHeatFlux(dayData);
  double slope = calculateSlope(dayData);
  double distanceSun = calculateDistanceSun(yday);
  double sunDeclination = calculateSunDeclination(yday);
  double omegaS = calculateOmegaS(sunDeclination, psi);
  double exTerRad = calculateExTerRad(distanceSun, omegaS, psi, sunDeclination);
  double clearSkyRad = calculateClearSkyRad(exTerRad, geoData);
  double netShortRad = calculateNetShortRad(solarRad);
  AbsTemp absTemp = calculateAbsTemp(dayData);
  double satVaP = calculateSatVaP(dayData);
  double actVaP = calculateActVaP(dayData, satVaP);
  double netLongRad = calculateNetLongRad(absTemp, actVaP, solarRad, clearSkyRad);
  double netRad = calculateNetRad(netShortRad, netLongRad);
  double airP = calculateAirP(geoData);
  double gamma = calculateGamma(latentVaporHeatFlux, airP);
  double etFao = calculateEtFao(netRad, dayData, satVaP, actVaP, slope, gamma);

  dayData.setEt0(etFao);

}


/**
 * Method to convert the geo position degree into rad.
 *
 * @author MD
 * @param geoData An object of the GeoData class
 * @return double geographical width position in rad aka. psi
 */
private static double convertDegToRad(GeoData geoData) {
  double psi = geoData.getGeoWid() * PI / 180;
  return (psi);
}


/**
 * Method to calcualte the latent vapor heat flux from temperature mean.
 *
 * @param dayData DayData class with mean temperature
 * @return double latent vapor heat flux L
 */
private static double calculateLatentVaporHeatFlux(DayData dayData) {
```

```
  return (2.501 - 0.002361 * dayData.getTempMean());
}

/**
 * Method to calculate the slope of vapor saturation curve.
 *
 * @param dayData DayData class with mean temperature
 * @return double s or delta
 */
private static double calculateSlope(DayData dayData) {
  double meanT = dayData.getTempMean();
  return ((4098 * 0.6108 * exp((17.27 * meanT) / (237.3 + meanT))) / pow((237.3 +
      meanT), 2));
}

/**
 * Method to calculate the relative distance of the earth to the sun.
 *
 * @param yday double day of the year 1-366
 * @return double returns the relative distance dr
 */
private static double calculateDistanceSun(double yday) {
  return (1 + 0.033 * cos((2 * PI / 365) * yday));
}

/**
 * Method to calculate the declination of the sun.
 *
 * @param yday double day of the year 1-366
 * @return double theta
 */
private static double calculateSunDeclination(double yday) {
  return (0.409 * sin((2 * PI / 365) * yday - 1.39));
}

/**
 * Method to calculate hour arc on sunset.
 *
 * @param theta declination of the sun
 * @param psi geo position in rad
 * @return
 */
private static double calculateOmegaS(double theta, double psi) {
  return (acos(-tan(psi) * tan(theta)));
}


/**
 * Method to calculate extra terrestrial radiation.
 *
 * @param GSC constant
 * @param distanceSun dr relative distance earth to sun
 * @param omegas hour arc on sunset
 * @param psi geo postion in rad
 * @param sunDeclination theta declination of the sun
 * @return double Ra exTerRad
```

```
 */
private static double calculateExTerRad(double distanceSun, double omegaS, double
    psi,
    double sunDeclination) {
  return ((24 * 60 / PI) * GSC * distanceSun
      * (omegaS * sin(psi) * sin(sunDeclination) + cos(psi) * cos(sunDeclination)
          * sin(omegaS)));
}

/*
 * Method to calculate global irradiation by cloudless sky angstroem coeff (0.25 +
    0.5)*Ra
 *
 * @param exTerRad extra terestrial radiation Ra
 *
 * @return cloudless sky global irradiatoin Rs0
 */
private static double calculateClearSkyRad(double exTerRad, GeoData geoData) {
  if (OLDRS0) {

    return (0.75 * exTerRad);
  } else {
    return ((0.75 + 2 * 10e-5 * geoData.getHeighAbvNn()) * exTerRad);
  }
}

/**
 * Method to calculate short wave net radiation.
 *
 * @param solarRad Rs measured solar radiation / global irritation
 * @return short wave net radiation Rns
 */
private static double calculateNetShortRad(double solarRad) {
  return ((1 - ALPHA) * solarRad);
}


/**
 * Method to calculate the absolute Temperature in kelvin.
 *
 * @param dayData class with meanTemp Temperature mean
 * @return absTemp absolut temperature values of class absTemp as output
 */
private static AbsTemp calculateAbsTemp(DayData dayData) {
  double meanTemp = dayData.getTempMean();
  double maxTemp = dayData.getTempMax();
  double minTemp = dayData.getTempMin();
  AbsTemp absTemp = new AbsTemp(0.0, 0.0, 0.0);
  absTemp.setAbsTempMean(273.16 + meanTemp);
  absTemp.setAbsTempMax(273.16 + maxTemp);
  absTemp.setAbsTempMin(273.16 + minTemp);
  return (absTemp);
}

/**
 * Template method to calculate saturation vapor pressure from Temp.
```

```
 *
 * @see calculateSatVP
 * @param DayData class with meanTemp Temperature mean
 * @return satVp
 */
private static double tempCalcSatVaP(double temp) {

  return (0.6108 * exp((17.27 * temp) / (237.3 + temp)));
}

/**
 * Method to calculate mean saturation vapor pressure.
 *
 * @param maxTemp Temperature maximum
 * @param minTemp Temperature minimum
 * @return es saturation vapor pressure
 */
private static double calculateSatVaP(DayData dayData) {
  if (dayData.getTempMax() != null && dayData.getTempMax() != null) {
    // b.c.
    // sane value
    double maxTemp = dayData.getTempMax();
    double minTemp = dayData.getTempMin();
    double maxVaP = tempCalcSatVaP(maxTemp);
    double minVaP = tempCalcSatVaP(minTemp);
    double meanVaP = (maxVaP + minVaP) / 2;
    return (meanVaP);
  } else {
    double meanTemp = dayData.getTempMean();
    return (tempCalcSatVaP(meanTemp));
  }
}

/**
 * Method to calculate the actual vapor pressure.
 *
 * @param dayData class containing airHumidityRelmean relative air humidity mean.
    Optional min and
 *        max TODO
 * @param satVP saturation vapor pressure @see calculateSatVP
 * @return actual Vapor pressure ea
 */
private static double calculateActVaP(DayData dayData, double satVaP) {
  double airHumidityRel = dayData.getAirHumidityRelMean();

  if (dayData.getAirHumidityRelMin() != null && dayData.getAirHumidityRelMax() !=
      null) {
    return ((tempCalcSatVaP(dayData.getTempMin()) * dayData.getAirHumidityRelMax()
        / 100
        + tempCalcSatVaP(dayData.getTempMax()) * dayData.getAirHumidityRelMin() /
            100) / 2);
  } else {
    return (satVaP * airHumidityRel / 100);
  }
}
```

```
/**
 * Method to calculate long wave net radiation. Depends on flag OLDRNL. If OLDRNL
    long wave net
 * radiation is calculated with temperature mean, else it is calculated with
    temperature minimum
 * and maximum.
 *
 * @param absTemp absolute Temperature
 * @param absTempMax absolute Temperature maximum
 * @param absTempMin absolute Temperature minimum
 * @param actVP actual vapor pressure
 * @param solarRad sRad global radiation / is measured
 * @param clearSkyRad sRad0 clear sky global radiation
 * @return NetLongRad long wave net radiation Rnl
 */
private static double calculateNetLongRad(AbsTemp absTemp, double actVaP, double
    solarRad,
    double clearSkyRad) {
  if (OLDRNL) {
    return (4.901e-09 * pow(absTemp.getAbsTempMean(), 4) * (0.34 - 0.14 *
        sqrt(actVaP))
        * (1.35 * (solarRad / clearSkyRad) - 0.35));
  } else {
    return (4.901e-09 * ((pow(absTemp.getAbsTempMax(), 4) +
        pow(absTemp.getAbsTempMin(), 4)) / 2)
        * (0.34 - 0.14 * sqrt(actVaP)) * (1.35 * (solarRad / clearSkyRad) - 0.35));
  }
}

/**
 * Method to calculate net radiation.
 *
 * @param netShortRad net short radiation
 * @param netLongRad net long radiation
 * @return netRad net radiation Rn
 */
private static double calculateNetRad(double netShortRad, double netLongRad) {
  return (netShortRad - netLongRad);
}


/**
 * Method to calculate the air pressure for the location.
 *
 * @param geoData class containing heighAbvNn location high above normal null
 * @return AirP air pressure P
 */
private static double calculateAirP(GeoData geoData) {

  return (101.3 * pow(((293 - 0.0065 * geoData.getHeighAbvNn()) / 293), 5.26));
}

/**
 * Method to calculate the psychrometric constant. Depends on OLDGAMMA flag. No
    obvious
```

```
     * differences between calculations. If OLDGAMMA then gamma is calculated with
     * latenetVapourHeutFlux otherwise not.
     *
     * @param latentVaporHeatFlux latent vapor heat flux
     * @param airP air pressure
     * @return gamma psychrometric constant
     */
    private static double calculateGamma(double latentVaporHeatFlux, double airP) {

      double gammaOld = ((0.001013 * airP) / (0.622 * latentVaporHeatFlux));
      double gammaNew = 0.665 * pow(10, -3) * airP;
      if (OLDGAMMA) {
        return (gammaOld);
      } else {
        return (gammaNew);
      }
    }


    private static double calculateEtFao(double netRad, DayData dayData, double
        satVaP, double actVaP,
        double slope, double gamma) {
      double meanTemp = dayData.getTempMean();
      double u2 = dayData.getWindspeed2m();
      return ((0.408 * slope * (netRad - G)
          + gamma * (900 / (meanTemp + 273) * u2 * (satVaP - actVaP)))
          / (slope + gamma * (1 + 0.34 * u2)));
    }

    /**
     * Method to extract the day of the year 1-366 from a date from a DayData object.
     *
     * @author MD
     * @param dayData DayData object
     * @return An Integer with the value of day of the year 1-366
     */
    private static Integer tellDoy(DayData dayData) {
      Calendar cal = Calendar.getInstance();
      cal.setTime(dayData.getDate());
      Integer yday = cal.get(Calendar.DAY_OF_YEAR);
      return (yday);
    }
}
```

B.4 GeoData.java

```
package de.hgu.gsehen.evapotranspiration;

/**
 * Class to represent the geographical data needed to calculate the et0.
 *
 * @author MD
 *
 */
public class GeoData {

  private boolean mean;
```

```
  private double geoLen;
  private double geoWid;
  private double heighAbvNn;

  public boolean isMean() {
    return mean;
  }

  public void setMean(boolean mean) {
    this.mean = mean;
  }

  public double getGeoLen() {
    return geoLen;
  }

  public void setGeoLen(double geoLen) {
    this.geoLen = geoLen;
  }

  public double getGeoWid() {
    return geoWid;
  }

  public void setGeoWid(double geoWid) {
    this.geoWid = geoWid;
  }

  public double getHeighAbvNn() {
    return heighAbvNn;
  }

  public void setHeighAbvNn(double heighAbvNn) {
    this.heighAbvNn = heighAbvNn;
  }

  /**
   * Constructor GeoData class. Typical values for Geisenheim:
   * geoLen=7.95,geoWid=49.99,heighAbvNn=110
   *
   * @param mean Calculation only with Temperature mean? TRUE = yes, FALSE = no
   * @param geoLen Geographical length of the location your are calculating et0 from
       in degree
   * @param geoWid Geographical width of the location your are calculating et0 from
       in degree
   * @param heighAbvNn Height over normal null in m
   */
  public GeoData(boolean mean, double geoLen, double geoWid, double heighAbvNn) {
    super();
    this.mean = mean;
    this.geoLen = geoLen;
    this.geoWid = geoWid;
    this.heighAbvNn = heighAbvNn;
  }
}
```

B.5 UtilityFunctions.java

```
package de.hgu.gsehen.evapotranspiration;

import static java.lang.Math.log;

public class UtilityFunctions {
  /**
   * Method to convert wind speed measurements to a height of 2 m above ground level.
   *
   * @param v Windspeed in m/s
   * @param hw Height of the anemometer in m
   * @return windspeed converted to 2 m above grund level
   */
  public static double convertWindSpeed2m(Double v, Double hw) {
    if (hw.equals(2.0)) {
      return (v);
    } else {
      return ((v * 4.87) / log(67.8 * hw - 5.42));
    }
  }
}
```

B.6 DailyBalance.java

```
package de.hgu.gsehen.gsbalance;

import java.time.LocalDate;
import java.time.ZoneId;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;

import de.hgu.gsehen.Gsehen;
import de.hgu.gsehen.evapotranspiration.DayData;
import de.hgu.gsehen.model.Crop;
import de.hgu.gsehen.model.CropDevelopmentStatus;
import de.hgu.gsehen.model.Plot;
import de.hgu.gsehen.model.SoilProfile;

public class DailyBalance {

  private static final Logger LOGGER = Logger.getLogger(Gsehen.class.getName());

  /**
   * Method to determine the current kc-value for the day and actual crop on the
      plot.
   *
   * @param dayData the day
   * @param plot the Plot
   */
  public static void determineCurrentKc(DayData dayData, Plot plot, SoilProfile
      soilProfile) {
    Date today = dayData.getDate();
    Date cropStart = plot.getCropStart();
    Date cropEnd = plot.getCropEnd();
    Date soilStart = plot.getSoilStartDate();
```

```
Crop crop = plot.getCrop();
CropDevelopmentStatus cropDevelopmentStatus = plot.getCropDevelopmentStatus();
Double kc1 = crop.getKc1();
Double kc2 = crop.getKc2();
Double kc3 = crop.getKc3();
Double kc4 = crop.getKc4();
Double soilKc;
if (soilProfile != null) {
  if (soilProfile.getSoilManualData().getSoilKc() != null) {
    soilKc = soilProfile.getSoilManualData().getSoilKc();
  } else {
    soilKc = 0.3;
    LOGGER.log(Level.INFO, "No soil kc: set to standard 0.3");
  }
} else {
  soilKc = 0.3;
  LOGGER.log(Level.INFO, "No soil kc: set to standard 0.3");
}
Double currentKc = null;
int phase1;
Integer phase2;
Integer phase3;
Integer phase4;
if (cropDevelopmentStatus.getPhase1() != null) {
  phase1 = cropDevelopmentStatus.getPhase1();
} else {
  phase1 = crop.getPhase1();
}

if (cropDevelopmentStatus.getPhase2() != null) {
  phase2 = cropDevelopmentStatus.getPhase2();
} else {
  phase2 = crop.getPhase2();
}

if (cropDevelopmentStatus.getPhase3() != null) {
  phase3 = cropDevelopmentStatus.getPhase3();
} else {
  phase3 = crop.getPhase3();
}
if (cropDevelopmentStatus.getPhase4() != null) {
  phase4 = cropDevelopmentStatus.getPhase4();
} else {
  phase4 = crop.getPhase4();
}

if (soilStart == null && cropStart == null) {
  currentKc = 0.0;
} else if (cropStart == null && today.compareTo(soilStart) >= 0) {
  currentKc = soilKc;
} else {
  if (cropEnd == null) {
    cropEnd = today;
  }

  if (soilStart != null) {
```

```
      if (today.compareTo(soilStart) >= 0 && today.compareTo(cropStart) < 0) {
        currentKc = soilKc;
      }
    }
    if (today.compareTo(cropStart) >= 0 && today.compareTo(cropEnd) <= 0) {
      LocalDate localToday =
          today.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
      LocalDate localCropStart =
          cropStart.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
      LocalDate localCropEnd =
          cropEnd.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
      if (localToday.compareTo(localCropStart.plusDays(phase1)) < 0
          || (localToday.compareTo(localCropEnd) <= 0 && phase2 == null)) {
        currentKc = kc1;
      } else if (localToday.compareTo(localCropStart.plusDays(phase1 + phase2)) < 0
          || (localToday.compareTo(localCropEnd) <= 0 && phase3 == null)) {
        if (kc2 == null) {
          throw new NullPointerException();
        }
        currentKc = kc2;
      } else if (localToday.compareTo(localCropStart.plusDays(phase1 + phase2 +
        phase3)) < 0
          || (localToday.compareTo(localCropEnd) <= 0 && phase4 == null)) {
        if (kc3 == null) {
          throw new NullPointerException();
        }
        currentKc = kc3;
      } else if (localToday
          .compareTo(localCropStart.plusDays(phase1 + phase2 + phase3 + phase4)) <
              0
          || localToday.compareTo(localCropEnd) <= 0) {
        if (kc4 == null) {
          throw new NullPointerException();
        }
        currentKc = kc4;
      }


    }
  }
  dayData.setCurrentKc(currentKc);

}



/**
 * Calculates the actual/current crop evapotranspiration.
 *
 * @param dayData the day
 * @param plot the plot
 */
public static void calculateEtc(DayData dayData, Plot plot) {
  // TODO: Add logging event
  Double et0 = dayData.getEt0();
  Double kc = dayData.getCurrentKc();
```

```
    Double faktor = plot.getScalingFactor();
    if (faktor == null) {
      faktor = 1.0;
    }
    dayData.setEtc(et0 * kc * faktor);
  }


  /**
   * Method to calculate the daily water balance.
   *
   * @param dayData the day
   * @throws IllegalStateException if prerequisites are not fullfilled
   */
  public static void calculateDailyBalance(DayData dayData) throws
      IllegalStateException {
    Double precipitation = dayData.getPrecipitation();
    if (precipitation == null) {
      throw new IllegalStateException("Precipitation has not been provided"); //
          logging
    }
    Double etc = dayData.getEtc();
    if (etc == null) {
      throw new IllegalStateException("Etc has not been calculated"); // logging
    }
    Double irrigation = dayData.getIrrigation();
    dayData.setDailyBalance(etc - precipitation - irrigation);

  }
}
```

B.7 DayDataCalculation.java

```
package de.hgu.gsehen.gsbalance;

//import de.hgu.gsehen.Gsehen;
//import de.hgu.gsehen.model.WeatherDataSource;
//import de.hgu.gsehen.util.PluginUtil;

public class DayDataCalculation {
  //  /**
  //   * Recalculates today's day data.
  //   */
  //  @SuppressWarnings({"checkstyle:rightcurly"})
  //  public void recalculateDayData() {
  //    for (WeatherDataSource weatherDataSource :
      Gsehen.getInstance().getWeatherDataSources()) {
  //      Gsehen.getInstance().sendDayDataChanged(
  //        PluginUtil.getPlugin(weatherDataSource.getPluginJsFileName()),
      weatherDataSource, null);
  //    }
  //  }
}
```

B.8 RecommendedAction.java

```java
package de.hgu.gsehen.gsbalance;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class RecommendedAction {

  @Id
  @GeneratedValue
  private long id;
  RecommendedActionEnum recommendation;
  Double availableWater;
  Double availableWaterPercent;
  Integer projectedDaysToIrrigation;
  private Double waterContentToAim;

  /**
   * Constructor for a recommended (irrigation) action.
   *
   * @param recommendation recommended action
   * @param availableWater The remaining available soil water
   * @param availableWaterPercent Percentual remaining available soil water
   * @param projectedDaysToIrrigation A simple projection of days until an
      irrigation might be
   *        recommended
   */
  public RecommendedAction(RecommendedActionEnum recommendation, Double
      availableWater,
      Double availableWaterPercent, Integer projectedDaysToIrrigation) {
    super();
    this.recommendation = recommendation;
    this.availableWater = availableWater;
    this.availableWaterPercent = availableWaterPercent;
    this.projectedDaysToIrrigation = projectedDaysToIrrigation;
  }

  public RecommendedAction() {}

  public RecommendedActionEnum getRecommendation() {
    return recommendation;
  }

  public void setRecommendation(RecommendedActionEnum recommendation) {
    this.recommendation = recommendation;
  }

  /**
   * Returns a value suited for java message parameter substitution.
   *
   * @param index the parameter index
   * @return the value of the appropriate property
   */
  public Object getParameterValue(int index) {
    switch (index) {
```

```
      case 0:
        return getAvailableWater();
      case 1:
        return getProjectedDaysToIrrigation();
      case 2:
        return getWaterContentToAim();
      default:
        return null;
    }
  }

  public Double getAvailableWater() {
    return availableWater;
  }

  public void setAvailableWater(Double availableWater) {
    this.availableWater = availableWater;
  }

  public Double getAvailableWaterPercent() {
    return availableWaterPercent;
  }

  public void setAvailableWaterPercent(Double availableWaterPercent) {
    this.availableWaterPercent = availableWaterPercent;
  }

  public Integer getProjectedDaysToIrrigation() {
    return projectedDaysToIrrigation;
  }

  public void setProjectedDaysToIrrigation(Integer projectedDaysToIrrigation) {
    this.projectedDaysToIrrigation = projectedDaysToIrrigation;
  }

  public Double getWaterContentToAim() {
    return waterContentToAim;
  }

  public void setWaterContentToAim(Double waterContentToAim) {
    this.waterContentToAim = waterContentToAim;
  }
}
```

B.9 RecommendedActionEnum.java

```
package de.hgu.gsehen.gsbalance;

public enum RecommendedActionEnum {
  EXCESS("gsbalance.recommended.action.EXCESS"),
  PAUSE("gsbalance.recommended.action.PAUSE"),
  SOON("gsbalance.recommended.action.SOON"),
  NOW("gsbalance.recommended.action.NOW"),
  IRRIGATION("gsbalance.recommended.action.IRRIGATION"),
  NO_DATA("gsbalance.recommended.action.NO_DATA"),
  TOMORROW("gsbalance.recommended.action.TOMORROW");
```

```
  private String messagePropertyKey;

  RecommendedActionEnum(String messagePropertyKey) {
    this.messagePropertyKey = messagePropertyKey;
  }

  public String getMessagePropertyKey() {
    return messagePropertyKey;
  }
}
```

B.10 Recommender.java

```
package de.hgu.gsehen.gsbalance;

import de.hgu.gsehen.Gsehen;
import de.hgu.gsehen.evapotranspiration.DayData;
import de.hgu.gsehen.evapotranspiration.EnvCalculator;
import de.hgu.gsehen.event.DayDataChanged;
import de.hgu.gsehen.event.GsehenEventListener;
import de.hgu.gsehen.event.ManualDataChanged;
import de.hgu.gsehen.model.Farm;
import de.hgu.gsehen.model.Field;
import de.hgu.gsehen.model.ManualData;
import de.hgu.gsehen.model.ManualWaterSupply;
import de.hgu.gsehen.model.Plot;
import de.hgu.gsehen.model.WaterBalance;
import de.hgu.gsehen.util.CollectionUtil;
import de.hgu.gsehen.util.DateUtil;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Recommender {
  private Gsehen gsehenInstance;

  {
    gsehenInstance = Gsehen.getInstance();
    gsehenInstance.registerForEvent(DayDataChanged.class,
        new GsehenEventListener<DayDataChanged>() {
          @Override
          public void handle(DayDataChanged event) {
            for (Farm farm : gsehenInstance.getFarmsList()) {
              for (Field field : farm.getFields()) {
                for (Plot plot : field.getPlots()) {
                  final DayData eventDayData = event.getDayData();
                  final Date eventDayDataDate =
                      eventDayData == null ? null : eventDayData.getDate();
                  if (event.isFromWeatherDataSource(field.getWeatherDataSourceUuid())
                      && DateUtil.between(eventDayDataDate,
                          CollectionUtil.nvl(plot.getSoilStartDate(),
                              plot.getCropStart()),
                          plot.getCropEnd())) {
                    copyWeatherData(eventDayData, getCurrentDayData(plot,
                        eventDayDataDate));
                  }
```

```
                performCalculations(field, plot);
              }
            }
          }
        }
      });
    gsehenInstance.registerForEvent(ManualDataChanged.class,
        event -> performCalculations(event.getField(), event.getPlot()));
  }

  private DayData getCurrentDayData(Plot plot, final Date eventDayDataDate) {
    if (plot == null) {
      throw new IllegalArgumentException("No plot given for day data calculation!");
    }
    guaranteeDailyBalances(plot);
    final List<DayData> plotDayDataList = plot.getWaterBalance().getDailyBalances();
    DayData plotCurrentDayData = null;
    for (DayData plotDayData : plotDayDataList) {
      if (eventDayDataDate.equals(plotDayData.getDate())) {
        plotCurrentDayData = plotDayData;
        break;
      }
    }
    if (plotCurrentDayData == null) {
      plotCurrentDayData = new DayData();
      plotCurrentDayData.setDate(eventDayDataDate);
      plotDayDataList.add(plotCurrentDayData);
    }
    return plotCurrentDayData;
  }

  private void applyManualData(DayData dayData, Plot plot) {
    if (plot == null) {
      throw new IllegalArgumentException("No plot given for manual data
          processing!");
    }
    guaranteeManualData(plot);
    final Date date = dayData.getDate();
    final List<ManualWaterSupply> manualList =
        plot.getManualData().getManualWaterSupply();
    double irrigation = 0.0;
    Double precipitation = null;
    for (ManualWaterSupply waterSupply : manualList) {
      if (date.equals(waterSupply.getDate())) {
        if (waterSupply.getIrrigation() != null) {
          irrigation = waterSupply.getIrrigation();
        }
        if (waterSupply.getPrecipitation() != null) {
          precipitation = waterSupply.getPrecipitation();
        }
        break;
      }
    }
    dayData.setIrrigation(irrigation);
    if (precipitation != null) {
      dayData.setPrecipitation(precipitation);
```

```
    }
  }

  private void performCalculations(Field field, Plot plot) {
    if (field == null || plot == null) {
      throw new IllegalArgumentException("No field or plot given for day data
          calculation!");
    }
    guaranteeDailyBalances(plot);
    for (DayData dayData : plot.getWaterBalance().getDailyBalances()) {
      applyManualData(dayData, plot);
      EnvCalculator.calculateEt0(dayData, gsehenInstance
          .getWeatherDataSourceForUuid(field.getWeatherDataSourceUuid()).getLocation());
      DailyBalance.determineCurrentKc(dayData, plot,
          gsehenInstance.getSoilProfileForUuid(field.getSoilProfileUuid()));
      DailyBalance.calculateEtc(dayData, plot);
      DailyBalance.calculateDailyBalance(dayData);
      TotalBalance.determineCurrentRootingZone(dayData, plot,
          gsehenInstance.getSoilProfileForUuid(field.getSoilProfileUuid()));
      TotalBalance.calculateCurrentAvailableSoilWater(dayData,
          gsehenInstance.getSoilProfileForUuid(field.getSoilProfileUuid()));
    }
    TotalBalance.calculateTotalWaterBalance(plot,
        gsehenInstance.getSoilProfileForUuid(field.getSoilProfileUuid()));
    TotalBalance.recommendIrrigation(plot);
    gsehenInstance.sendRecommendedActionChanged(plot, null);
  }

  private void copyWeatherData(DayData eventDayData, DayData plotCurrentDayData) {
    plotCurrentDayData.setTempMax(eventDayData.getTempMax());
    plotCurrentDayData.setTempMin(eventDayData.getTempMin());
    plotCurrentDayData.setTempMean(eventDayData.getTempMean());
    plotCurrentDayData.setAirHumidityRelMax(eventDayData.getAirHumidityRelMax());
    plotCurrentDayData.setAirHumidityRelMin(eventDayData.getAirHumidityRelMin());
    plotCurrentDayData.setAirHumidityRelMean(eventDayData.getAirHumidityRelMean());
    plotCurrentDayData.setGlobalRad(eventDayData.getGlobalRad());
    plotCurrentDayData.setPrecipitation(eventDayData.getPrecipitation());
    plotCurrentDayData.setWindspeed2m(eventDayData.getWindspeed2m());
  }

  private void guaranteeDailyBalances(Plot plot) {
    if (plot.getWaterBalance() == null) {
      plot.setWaterBalance(new WaterBalance());
    }
    if (plot.getWaterBalance().getDailyBalances() == null) {
      plot.getWaterBalance().setDailyBalances(new ArrayList<DayData>());
    }
  }

  private void guaranteeManualData(Plot plot) {
    if (plot.getManualData() == null) {
      plot.setManualData(new ManualData());
    }
    if (plot.getManualData().getManualWaterSupply() == null) {
      plot.getManualData().setManualWaterSupply(new ArrayList<ManualWaterSupply>());
    }
```

```
  }
}
```

B.11 TotalBalance.java

```
package de.hgu.gsehen.gsbalance;

import java.time.LocalDate;
import java.time.ZoneId;
import java.util.Date;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

import de.hgu.gsehen.Gsehen;
import de.hgu.gsehen.evapotranspiration.DayData;
import de.hgu.gsehen.model.Crop;
import de.hgu.gsehen.model.CropDevelopmentStatus;
import de.hgu.gsehen.model.CropRootingZone;
import de.hgu.gsehen.model.Plot;
import de.hgu.gsehen.model.Soil;
import de.hgu.gsehen.model.SoilManualData;
import de.hgu.gsehen.model.SoilProfile;
import de.hgu.gsehen.model.SoilProfileDepth;

public class TotalBalance {

  private static final Logger LOGGER = Logger.getLogger(Gsehen.class.getName());

  @SuppressWarnings("checkstyle:javadocmethod")
  public static void determineCurrentRootingZone(DayData dayData, Plot plot,
      SoilProfile soilProfile) {
    final Date today = dayData.getDate();
    final Date cropStart = plot.getCropStart();
    Date cropEnd = plot.getCropEnd();
    final Date soilStart = plot.getSoilStartDate();
    Crop crop = plot.getCrop();
    CropRootingZone cropRootingZone = plot.getCropRootingZone();
    Integer rootingZone1;
    Integer rootingZone2;
    Integer rootingZone3;
    Integer rootingZone4;
    if (cropRootingZone.getRootingZone1() != null) {
      rootingZone1 = cropRootingZone.getRootingZone1();
    } else {
      rootingZone1 = crop.getRootingZone1();
    }

    if (cropRootingZone.getRootingZone2() != null) {
      rootingZone2 = cropRootingZone.getRootingZone2();
    } else {
      rootingZone2 = crop.getRootingZone2();
    }

    if (cropRootingZone.getRootingZone3() != null) {
      rootingZone3 = cropRootingZone.getRootingZone3();
    } else {
```

```
    rootingZone3 = crop.getRootingZone3();
  }

  if (cropRootingZone.getRootingZone4() != null) {
    rootingZone4 = cropRootingZone.getRootingZone4();
  } else {
    rootingZone4 = crop.getRootingZone4();
  }
  SoilManualData soilManualData = soilProfile.getSoilManualData();
  Integer soilZone;
  if (soilManualData != null) {
    if (soilManualData.getSoilZone() != null) {
      soilZone = soilManualData.getSoilZone();
    } else {
      soilZone = 10;
      LOGGER.log(Level.INFO, "No soil zone: set to standard 10cm");
    }
  } else {
    soilZone = 10;
    LOGGER.log(Level.INFO, "No soil zone: set to standard 10cm");
  }
  CropDevelopmentStatus cropDevelopmentStatus = plot.getCropDevelopmentStatus();
  Integer currentRootingZone = null;
  int phase1;
  Integer phase2;
  Integer phase3;
  Integer phase4;
  if (cropDevelopmentStatus.getPhase1() != null) {
    phase1 = cropDevelopmentStatus.getPhase1();
  } else {
    phase1 = crop.getPhase1();
  }

  if (cropDevelopmentStatus.getPhase2() != null) {
    phase2 = cropDevelopmentStatus.getPhase2();
  } else {
    phase2 = crop.getPhase2();
  }

  if (cropDevelopmentStatus.getPhase3() != null) {
    phase3 = cropDevelopmentStatus.getPhase3();
  } else {
    phase3 = crop.getPhase3();
  }
  if (cropDevelopmentStatus.getPhase4() != null) {
    phase4 = cropDevelopmentStatus.getPhase4();
  } else {
    phase4 = crop.getPhase4();
  }

  if (soilStart == null && cropStart == null) {
    currentRootingZone = 0;
  } else if (cropStart == null && today.compareTo(soilStart) >= 0) {
    currentRootingZone = soilZone;
  } else if (soilStart != null) {
    if (cropEnd == null) {
```

```
        cropEnd = today;
      }
      if (today.compareTo(soilStart) >= 0 && today.compareTo(cropStart) < 0) {
        currentRootingZone = soilZone;
      }
      if (today.compareTo(cropStart) >= 0 && today.compareTo(cropEnd) <= 0) {
        LocalDate localToday =
            today.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
        LocalDate localCropStart =
            cropStart.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
        LocalDate localCropEnd =
            cropEnd.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
        if (localToday.compareTo(localCropStart.plusDays(phase1)) < 0
            || (localToday.compareTo(localCropEnd) <= 0 && phase2 == null)) {
          currentRootingZone = rootingZone1;
        } else if (localToday.compareTo(localCropStart.plusDays(phase1 + phase2)) < 0
            || (localToday.compareTo(localCropEnd) <= 0 && phase3 == null)) {
          if (rootingZone2 == null) {
            throw new NullPointerException();
          }
          currentRootingZone = rootingZone2;
        } else if (localToday.compareTo(localCropStart.plusDays(phase1 + phase2 +
            phase3)) < 0
            || (localToday.compareTo(localCropEnd) <= 0 && phase4 == null)) {
          if (rootingZone3 == null) {
            throw new NullPointerException();
          }
          currentRootingZone = rootingZone3;
        } else if (localToday
            .compareTo(localCropStart.plusDays(phase1 + phase2 + phase3 + phase4)) <
                0
            || localToday.compareTo(localCropEnd) <= 0) {
          if (rootingZone4 == null) {
            throw new NullPointerException();
          }
          currentRootingZone = rootingZone4;
        }


      }
    }
    Integer maxRootingZone = plot.getRootingZone();
    if (maxRootingZone != null) {
      if (currentRootingZone > maxRootingZone) {
        currentRootingZone = maxRootingZone;
        LOGGER.log(Level.INFO, "currentRootingZone set to maxRootingZone");
      }
    }
    dayData.setCurrentRootingZone(currentRootingZone);

  }

  @SuppressWarnings("checkstyle:javadocmethod")
  public static void calculateCurrentAvailableSoilWater(DayData dayData, SoilProfile
      soilProfile) {
    Integer currentRootingZone = dayData.getCurrentRootingZone();
```

```
    Double currentAvailableSoilWater = 0.0;
    Integer remaningRootingZone = currentRootingZone;
    List<Soil> soils = soilProfile.getSoilType();
    List<SoilProfileDepth> profiles = soilProfile.getProfileDepth();
    int i;
    for (i = 0; i < soils.size(); i++) {
      Double rest = (profiles.get(i).getDepth() - remaningRootingZone);
      if (rest >= 0 || rest < 0 && i + 1 == soils.size()) {
        currentAvailableSoilWater += remaningRootingZone * 100.0 * 100.0
            * (soils.get(i).getAvailableWaterCapacity() / 100) / 1000;
        break;
      } else if (rest < 0) {
        currentAvailableSoilWater += (remaningRootingZone + rest) * 100.0 * 100.0
            * (soils.get(i).getAvailableWaterCapacity() / 100) / 1000;
        remaningRootingZone = Math.abs(rest.intValue());
      }
    }
    dayData.setCurrentAvailableSoilWater(currentAvailableSoilWater);
  }

  // TODO: Starkregenereignis und dauer der Pause konfigurierbar machen.
  /**
   * Method to calculate the total water balance of a plot.
   *
   * @param plot the desired plot
   */
  public static void calculateTotalWaterBalance(Plot plot, SoilProfile soilProfile) {
    SoilManualData soilManualData = soilProfile.getSoilManualData();
    Double rainMax;
    Integer daysPause;
    if (soilManualData != null) {
      if (soilManualData.getRainMax() != null) {
        rainMax = soilManualData.getRainMax();
      } else {
        rainMax = 30.0;
        LOGGER.log(Level.INFO, "MaxRain event set to 30mm");
      }
    } else {
      rainMax = 30.0;
      LOGGER.log(Level.INFO, "MaxRain event set to 30mm");
    }
    if (soilManualData != null) {
      if (soilManualData.getDaysPause() != null) {
        daysPause = soilManualData.getDaysPause();
      } else {
        daysPause = 2;
        LOGGER.log(Level.INFO, "Days Pause set to 2");
      }
    } else {
      daysPause = 2;
      LOGGER.log(Level.INFO, "Days Pause set to 2");
    }
    Double startValue;
    List<DayData> dailyBalances = plot.getWaterBalance().getDailyBalances();
    if (dailyBalances.isEmpty()) {
      return;
```

```
    }
    if (plot.getSoilStartValue() != null) {
      startValue =
          dailyBalances.get(0).getCurrentAvailableSoilWater() *
              (plot.getSoilStartValue() / 100);
    } else {
      startValue = dailyBalances.get(0).getCurrentAvailableSoilWater();
    }
    int i;
    int size = dailyBalances.size();
    for (i = 0; i < size; i++) {
      plot.setCalculationPaused(false);
      Double currentTotalWaterBalance = null;
      if (i == 0) {
        currentTotalWaterBalance = startValue -
            dailyBalances.get(0).getDailyBalance();
        dailyBalances.get(0).setCurrentTotalWaterBalance(currentTotalWaterBalance);
      } else {
        currentTotalWaterBalance = dailyBalances.get(i -
            1).getCurrentTotalWaterBalance()
            - dailyBalances.get(i).getDailyBalance();
        dailyBalances.get(i).setCurrentTotalWaterBalance(currentTotalWaterBalance);
      }
      // Calculation Pause
      int k = 0;
      if (currentTotalWaterBalance >
          dailyBalances.get(i).getCurrentAvailableSoilWater()
          && dailyBalances.get(i).getPrecipitation() > rainMax) {
        plot.setCalculationPaused(true);
        for (k = 1; k <= daysPause; k++) {
          if (i + k > size) {
            break;
          }
          Double maxWater = dailyBalances.get(i + k).getCurrentAvailableSoilWater();
          dailyBalances.get(i + k).setCurrentTotalWaterBalance(maxWater);

        }

      }
      // TotalWaterBalance not bigger than CurrentAvailableSoilWater
      dailyBalances.get(i)
          .setCurrentTotalWaterBalance(Math.min(dailyBalances.get(i).getCurrentTotalWaterBalance()
              dailyBalances.get(i).getCurrentAvailableSoilWater()));

      if (k != 0 && k > 0) {
        i += k - 1;
      }
    }
  }

  /**
   * Method to recommend an irrigation decision for a plot.
   *
   * @param plot Plot in question
   */
  public static void recommendIrrigation(Plot plot) {
```

```
      RecommendedAction recommendedAction = new RecommendedAction(null, null, null,
          null);
      if (plot.getWaterBalance().getDailyBalances().isEmpty()) {
        recommendedAction.setRecommendation(RecommendedActionEnum.NO_DATA);
      } else {
        DayData currentDay = plot.getWaterBalance().getDailyBalances()
            .get(plot.getWaterBalance().getDailyBalances().size() - 1);
        Double currentAvailableSoilWater = currentDay.getCurrentAvailableSoilWater();
        Double waterContentToAim =
            currentAvailableSoilWater * 0.9 - currentDay.getCurrentTotalWaterBalance();
        Double availableWater = currentAvailableSoilWater * 0.3 - waterContentToAim;
        recommendedAction.setAvailableWater(availableWater);
        recommendedAction
            .setAvailableWaterPercent((availableWater / (currentAvailableSoilWater *
                0.3)) * 100);
        recommendedAction.setAvailableWater(availableWater);
        final int projectedDaysToIrrigation =
            Math.abs((int) Math.floor(availableWater / currentDay.getEtc()));
        recommendedAction.setProjectedDaysToIrrigation(projectedDaysToIrrigation);
        recommendedAction.setWaterContentToAim(waterContentToAim);

        if (waterContentToAim > currentAvailableSoilWater) {
          recommendedAction.setRecommendation(RecommendedActionEnum.EXCESS);
          // throw new UnsupportedOperationException(
          // "The water balance exceeds the total available soil water\n"
          // + "- your plants are dead for sure \\u2620");
        } else {
          if (plot.getCalculationPaused()) {
            recommendedAction.setRecommendation(RecommendedActionEnum.PAUSE);
          } else {
            if (availableWater > 0) {
              if (projectedDaysToIrrigation == 0) {
                recommendedAction.setRecommendation(RecommendedActionEnum.NOW);
              } else if (projectedDaysToIrrigation == 1) {
                recommendedAction.setRecommendation(RecommendedActionEnum.TOMORROW);
              } else {
                recommendedAction.setRecommendation(RecommendedActionEnum.SOON);
              }
            } else {
              recommendedAction.setRecommendation(RecommendedActionEnum.IRRIGATION);
            }
          }
        }
      }
      plot.setRecommendedAction(recommendedAction);
    }
  }
```