universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„SunScreen: Visual Fault Detection and Diagnosis for Solar-Thermal Systems“

verfasst von / submitted by

Lukas Feierl, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 910
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Computational Science UG2002
Betreut von / Supervisor:	Univ.-Prof. Torsten Möller, PhD
Mitbetreut von / Co-Supervisor:	

Abstract (English)

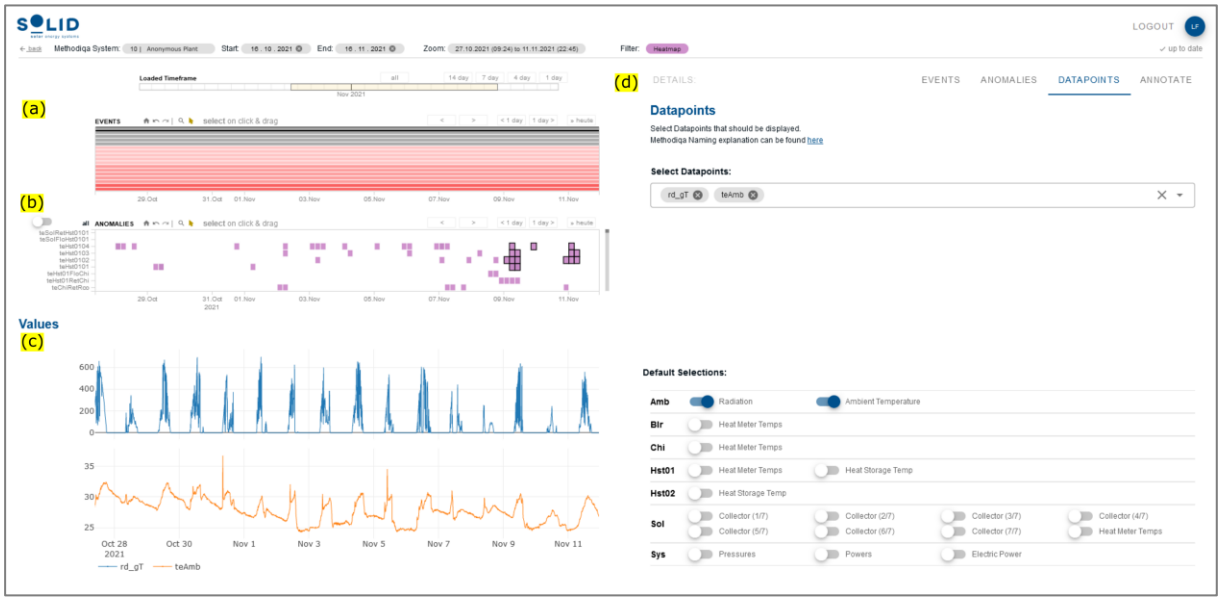


Figure 1: This work introduces the visual framework SunScreen*, which supports Fault Detection and Diagnosis for solar thermal systems. It consists of four main parts: (a) shows manually annotated system events (e.g., services or faults) as a Gantt chart. (b) shows detected anomalies for each sensor as a heatmap, and (c) shows selected measurement values as a time series. Users can access more details about data points, events, anomalies, and annotations in the Details Panel (d).*

Fault Detection and Diagnosis (FDD) are essential tasks for operating and maintaining solar-thermal systems. It allows operators to react to faults and ensures that the system runs at optimal performance. Thus, monitoring personnel frequently analyze the sensor data to detect any unusual behavior. However, this task is challenging due to the multivariate and time-dependent nature of the sensor data and the large number of sensors. As a result, considerable time and training are needed to investigate the system manually, while automatic FDD algorithms are often too sensitive or do not cover all critical faults.

To bridge the gap between manual and automatic approaches, this work introduces the visual framework *SunScreen*. It tries to combine the users' domain knowledge with automatic fault detection results and focuses explicitly on solar thermal data. Using this framework, the users can exploratively evaluate anomalies and analyze the complex sensor data. *SunScreen* is designed using Rapid-Prototyping and assessed in a Use-Case and Usability-Test, working closely with domain users from *SOLID Solar Energy Systems GmbH*. The results are promising, but *SunScreen* needs further refinement before it can fully support the tasks of the monitoring personnel. Nevertheless, the domain users appreciate that algorithm results and system-event information (e.g., already detected faults, services) are accessible during diagnosis.

The main contributions of this work are: (i) a detailed requirement analysis concerning FDD at solar thermal systems, (ii) the design and evaluation of *SunScreen*, (iii) a discussion of why the current version of *SunScreen* does not meet some requirements and how to improve and (iv) a reflection on using data comics to present the design of the visual framework.

Abstract (Deutsch)

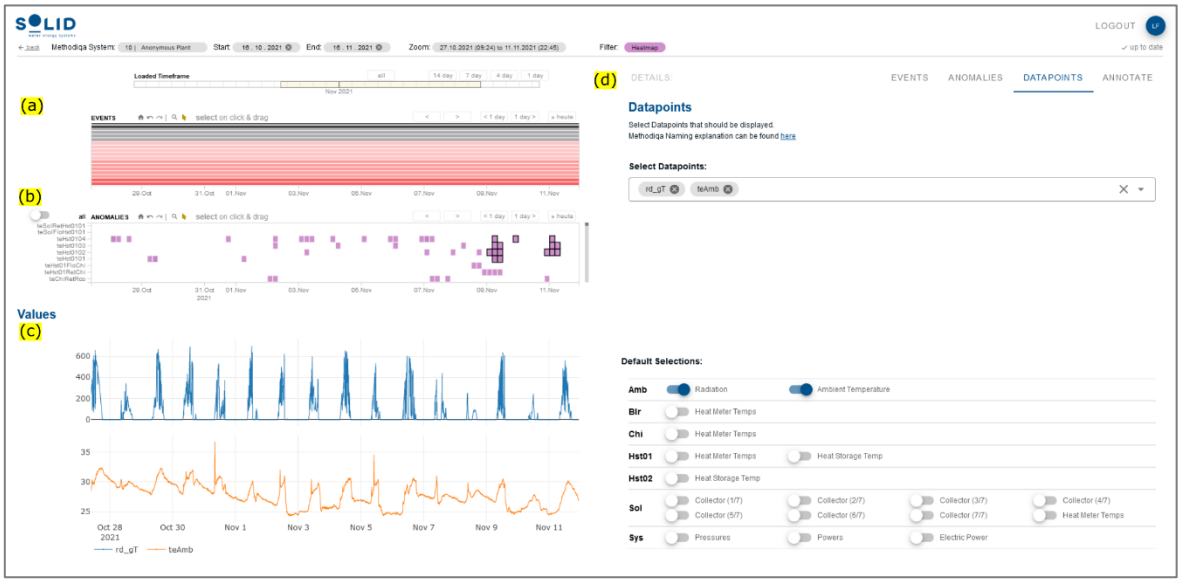


Figure 2: In dieser Masterarbeit wird SunScreen *vorgestellt - ein visuelles Interface zur Fehlererkennung und Diagnose bei solarthermischen Anlagen. Es besteht aus vier Hauptbereichen: (a) zeigt alle Ereignisse, die die Anlage zu einer gewissen Zeit beeinflussen (z.B.: Services, oder Fehler) als Gantt-Chart an. (b) zeigt Anomalien an, die von Fehler-Algorithmen erkannt wurden. Die Messwerte ausgewählter Sensoren können in (c) als Zeitreihe angezeigt werden. Im Detail-Panel (d) können mehr Informationen zu den Anlagen-Ereignissen, Anomalien, Messwerten und Annotationen angezeigt werden.*

Fehlererkennung und Diagnose ist sehr wichtig, um solarthermische Anlagen zu betreiben und überwachen zu können. Es erlaubt den Betreibern, auf Fehler zu reagieren und ihre Anlagen in einem guten Zustand zu halten. Daher analysiert das Monitoring-Personal regelmäßig die Anlagendaten, um Auffälligkeiten schnell zu erkennen. Diese Arbeit ist allerdings herausfordernd, da die Daten aufgrund des dynamischen Anlagenverhaltens zeitabhängig und multidimensional sind, und somit schwer zu interpretieren sind. Daher braucht es einige Zeit und Übung, um thermische Solaranlagen manuell zu überwachen, während Fehleralgorithmen öfters Fehlalarme auslösen oder wichtige Fehler übersehen.

Daher präsentiert diese Arbeit das visuelle Framework *SunScreen,* um das Monitoring-Personal in ihrer Aufgabe zu unterstützen. Es kombiniert die automatische Fehlererkennung mit dem Fachwissen des Monitoring-Personals und erlaubt es, mit den speziellen solarthermischen Daten umzugehen. Die Benutzer können die erkannten Anomalien interaktiv überprüfen und die dazugehörigen Sensor-Daten visualisieren. Das Design von *SunScreen* wurde mit Rapid-Prototyping entwickelt und mit Hilfe von Fachleuten von *SOLID Solar Energy Systems GmbH* in einem Usability Test und einem Use Case evaluiert. Die Ergebnisse zeigen, dass noch Verbesserungen notwendig sind, um das Monitoring-Personal voll zu unterstützen. Aber es zeigt sich auch das Potential von *SunScreen*: Zum Beispiel schätzen die Nutzer sehr, dass Anlagen-Ereignisse (z.B.: erkannte Fehler, Services) und detektierte Anomalien während der Messdaten-Analyse angezeigt werden können.

Die Hauptbeiträge dieser Arbeit sind: (i) eine detaillierte Anforderungsanalyse für Fehlererkennung und Diagnose für solarthermische Anlagen, (ii) das Design und die Validierung von *SunScreen*, (iii) eine Diskussion, warum die derzeitige Version von *SunScreen* noch nicht alle Anforderungen erfüllen kann und wie sie verbessert werden könnte und (iv) eine Reflexion über die Verwendung von Data-Comics, um das Design von *SunScreen* zu präsentieren.

Acknowledgments

My special thanks go to Torsten and Peter for their outstanding supervision and valuable feedback throughout this project. I would also like to thank Andreas, Christian, Franz, Marcel, Nico, and especially Hannes, Maria, and Patrick from SOLID for participating in multiple evaluation sessions and Viktor, Thomas, and Bernhard for vivid discussions about fault detection at solar thermal systems. In addition, I would like to thank my family, especially Isabella, for the emotional support during this time.

Content

1 Motivation

Fault Detection and Diagnosis (FDD) is essential for operating and maintaining solar-thermal systems. It allows operators to react to faults and ensures that the system runs at optimal performance. Thus, monitoring personnel frequently analyze the sensor data to detect unusual behavior. However, with increasing numbers of sensors and more complex systems, manually investigating the system data requires considerable time and training. To speed up the process, automatic fault detection algorithms can be used. Yet, it is difficult to create algorithms that perfectly cover all potential faults. For example, very sensitive algorithms are prone to raise false alarms, while insensitive algorithms will likely miss essential faults. Therefore, the monitoring personnel must often evaluate algorithms and manually diagnose faults.

Because these tasks are very explorative and a lot of information must be processed, visual frameworks can greatly support Fault Detection and Diagnosis. This way, the monitoring personnel can interactively explore the sensor data and algorithm results, combining domain knowledge and automatic fault detection.

Nevertheless, the main challenge for FDD at solar thermal plants is the time-dependent and multi-variate nature of the sensor data. For instance, consumer demand and ambient conditions like irradiation can be subject to rapid and drastic changes, influencing the system. Consequently, measurements must often be analyzed in dependence of these variations. Correlations between measurements are multidimensional (e.g., irradiation, ambient temperature, volume flow, return temperature, and shading from adjacent structures influencing the collector temperature), non-linear (e.g., pumps suddenly starting), and time-dependent (e.g., temperatures rising due to accumulated heat). Thus, visualization approaches must cope with all of these challenges to allow efficient evaluation and diagnosis of detected faults.

While some applications focusing on visual fault detection and diagnosis already exist, none can deal with all aspects above. For example, supervisory control and data acquisition (SCADA) software [1, 2, 3] provides graphical user interfaces to check the system's current status, supports the creation of simple alarms, and shows basic charts of the sensor data. However, they do not support analyzing multi-dimensional correlations of the measurements and provide little functionality for evaluating automatic fault detection results. Other examples are related to *anomaly detection*. This topic is almost identical to FDD, as faults usually cause abnormal system behavior. Thus, detecting anomalies and detecting faults often coincide. However, approaches so far either focus on entirely different data like networks [4, 5, 6] or traffic [7], or do not support the detailed analysis of multivariate dependencies [8, 9, 10, 11, 12, 13].

Hence, this work introduces ***SunScreen***, a visual prototype supporting fault detection and diagnosis for solar thermal systems. It combines the manual and automatic FDD approach by allowing users to evaluate detected anomalies and supports analyzing the corresponding measurement data. In addition, it allows the user to access manually annotated contextual information (e.g., services, known faults, notes from other users) to ease the Fault Detection and Diagnosis process.

The main contributions of this work are:

i. A requirement analysis for visual Fault Detection and Diagnosis at solar-thermal systems, describing the data and tasks related to the topic.
ii. The design and evaluation of the visual framework *SunScreen*.
iii. A discussion of why the current version of *SunScreen* does not meet all identified requirements and how to improve.
iv. A discussion on the usage of Data-Comics [14] to present the design of *SunScreen*.

The work is structured as follows: Chapter 2 discusses the methodology for developing and evaluating *SunScreen*. Chapter 3 gives a short introduction to the terminology used within this work. It also describes a new characterization for analyzing visual frameworks dealing with FDD. In chapter 4, the results of the requirement analysis are presented. It shows what kind of data is used for FDD at solar thermal plants and which tasks need to be solved by the monitoring personnel. Next, chapter 5 discusses related work and analyzes whether the requirements are met. Chapter 6 then presents *SunScreen* and justifies why certain design decisions were made. Chapter 7 describes the results of the Usability-Test and the Use-Case and provides a walkthrough of *SunScreen*. Finally, chapter 8 discusses encountered pitfalls, potential future work, and the usage of Data-Comics for presenting *SunScreen*.

2 Methodology

SunScreen was developed following the nine-stage design study framework from Sedlmair et al. [15] and design guidelines from Munzner [16]. A timeline showing the development process and important sessions with domain users can be seen in Figure 3.

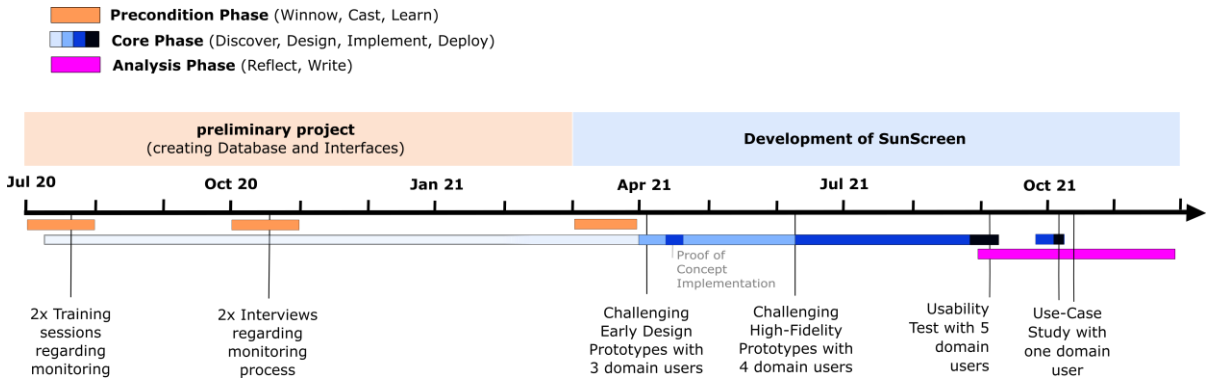


Figure 3: Timeline of the development of SunScreen *showing which design stages have been performed at which time. Important events with domain users are displayed as lines.*

Domain-specific information was gathered from real-life users by working closely together with the company *SOLID Solar Energy Systems GmbH*. As an operator of solar thermal systems worldwide, they monitor many large-scale plants and kindly agreed to provide real-life data and participate in multiple feedback sessions. Additional domain knowledge was acquired by working at the company part-time for four years before the project started.

Preliminary Project

In the early precondition phase of the project, multiple short meetings with a domain user were carried out to discover topics of interest. However, before developing *SunScreen*, a preliminary project was conducted to ensure sufficient data quality and availability.

During this time, database systems were added and modified while gathering more information about monitoring. As such, two interviews about the process in general (1 hour each) were held, and the author participated in two training sessions for performing monitoring (7 hours each). In addition, task and data requirements were analyzed during two contextual inquiry sessions analyzing a target user during his work (7 hours each). The results have been further validated in subsequent sessions with other participants and through comparison with related work.

Development of SunScreen

The design of *SunScreen* was developed using rapid prototyping, with feedback sessions at two

different levels of maturity. Three to four domain users (including one target user) were asked to provide feedback on three designs for each maturity level. New prototypes were then created based on the results - merging and modifying the designs accordingly. In addition, participants were also asked to provide feedback on the identified tasks and data characterizations. Based on the feedback, the prototypes were merged, and the first version of *SunScreen* was implemented.
SunScreen's usability was validated with five domain users (including one target user) using Usage Scenarios: After a short introduction of *SunScreen*, participants were asked to either perform predefined tasks or explore the tool on their own. Similar to contextual inquiries, they were asked to describe their interaction with *SunScreen*. Support was provided in case they needed help interacting with the system. After 1 hour, they were asked to fill out a System-Usability-Scale [17, 18, 19] questionnaire to provide feedback on the usability of *SunScreen*.
In addition, one domain user was asked to provide feedback in a Use-Case study. In two sessions, the participant was tasked to perform fault detection and diagnosis for one of SOLID's plants for two arbitrary months in the dataset. Similar to the Usage-Scenarios, the domain user was asked to describe his interaction with *SunScreen*. Following a more elaborate introduction to *SunScreen* (about one hour long), the participant was tasked to find as many faults as possible in about half an hour. This time limit was chosen to mimic the time constraints of actual monitoring sessions, where many systems must be checked as fast and thoroughly as possible. A semi-structured interview followed each session to discuss the results (about 30 minutes each).

3 Terminology

This section contains the terminology that is used within this work. It first presents definitions regarding fault detection before focusing on a classification for fault detection methods. In the last part of this section, a new custom characterization is introduced, which allows categorizing visual frameworks based on their Fault Detection and Diagnosis strategy.

3.1 Fault Detection and Diagnosis

Fault Detection and Diagnosis (FDD) is crucial for many domains in the industry, and thus definitions have been established. For example, Isermann and Ballé [20] proposed definitions regarding the Fault Management process. At the same time, the British Standard Institution provides similar definitions regarding monitoring industrial systems in the European norm *BS EN-13306:2017* [21].

Term	**Definition**
Fault	An unpermitted deviation of at least one characteristic property or parameter of the system from the acceptable/usual/standard condition.
Fault-Detection	Determination of the faults present in a system and the time of detection.
Fault-Diagnosis	Determination of the kind, size, location, and time of detection of a fault. Follows fault detection. Includes fault isolation and identification.
Fault isolation	Determination of the kind, location, and time of detection of a fault. Follows fault detection.
Fault identification	Determination of the size and time-variant behavior of a fault. Follows fault isolation.
Monitoring	A continuous real-time task of determining the conditions of a physical system, by recording information, recognizing, and indicating anomalies in the behavior.

Table 1: Terminology regarding Fault-Management from Isermann and Ballé [20].

This work uses definitions from Isermann and Ballé (see Table 1). The definitions show that Fault Detection and Diagnosis are essential to monitoring. In addition, they also clarify that fault detection has many similarities to the task of anomaly detection. This topic is almost identical to FDD, as faults usually cause abnormal system behavior. However, it is not safe to assume that every anomaly is a fault. Events like repairs, tests, or rare operating conditions can also cause anomalies in the data without being faults.

3.2 FDD Methods

Isermann and Ballé [20] also provide a categorization scheme for algorithms that can be used to identify and diagnose faults. They distinguish between quantitative and qualitative methods based on what apriori knowledge is used for detecting faults. Here, quantitative model-based methods use mathematical models to mimic the system's behavior, for example, using differential equations to describe the physical process of the system. This way, predictions for sensor values can be computed and compared with measurements to detect faults. In contrast, qualitative model-based methods use rule-based methods to detect faults. A simple example would be an if-then check whether a collector temperature is above a certain threshold.

The categorization from Isermann and Ballé has further been extended by Ventakatsubraminian et al. [22], adding a third category for process-history-based methods (see Figure 4). This category includes methods that use historical data to detect faults, such as machine learning algorithms or statistics. Thus, they distinguish between methods that use data-driven approaches (process-history-based) and methods derived by using domain knowledge (qualitative model-based and quantitative model-based).

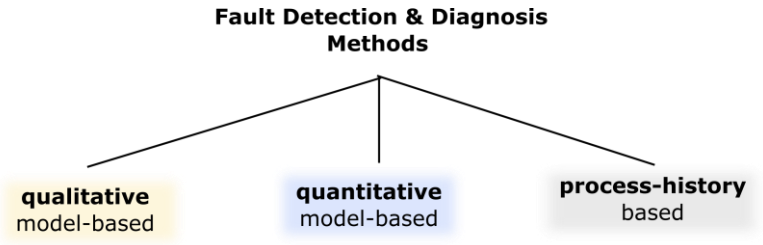


Figure 4: Categorization of Fault Detection and Diagnosis methods from Ventakatsubraminian et al. [22]. Only the first level of the hierarchy is shown.

3.3 FDD Strategies in VIS

The characterization above is focused on what apriori knowledge is used to formulate FDD algorithms. However, when analyzing visual frameworks, it might make more sense to group frameworks based on what data they display and how they display it instead.

To illustrate this, imagine an artificial neuronal network (process-history-based) and a physical-process-based algorithm (quantitative-model-based) that can predict the measurement values of a sensor perfectly. In this case, both algorithms yield the same predicted values. Thus, the same visual framework might be used to display the data, independently of which of the two algorithms was used to derive the predicted values. Therefore, a distinction on apriori knowledge makes less sense for categorizing visual frameworks, as this does not strongly influence the design of visual frameworks. Instead, this work introduces a custom characterization for FDD strategies in visualization that focuses on what data is used rather than how the data was derived or visualized (see Figure 5). Thus, the following categories distinguish whether the user has to rely on raw sensor data only or can access meta-data like predicted values, for example. The strategies are inspired by Huovinen and Hietanen [23] and Jenetzko et al. [13].

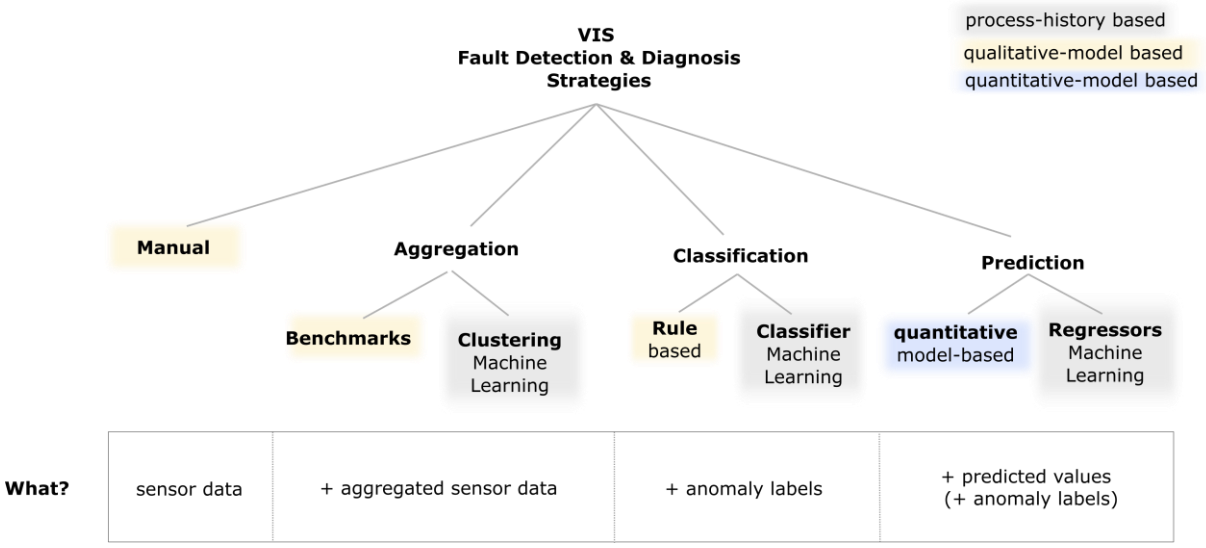


Figure 5: Custom categorization of VIS strategies for fault detection used in this work.

Manual. No additional information is extracted from the original sensor data. Instead, the users must manually analyze the raw sensor data themselves. Nevertheless, they might rely on visualization to show the data in the proper context, such as displaying line charts to visualize the measurement data. The key aspect of this approach is that no additional meta-data is generated.

Aggregation. The data is aggregated so the user can focus on a smaller part of the data. For example, benchmarks of the system can be computed to evaluate the system's status, or clustering approaches can be used to group similar data. The key aspect of this approach is that new information is generated, which summarizes the data. Thus, the user needs to view less information to evaluate the system status. However, these methods do not assess the anomaly of the data, such that users still have to check for outlier or anomalous groups to identify faults.

Classification. This strategy uses classification results to support the user during monitoring. These fault/no-fault classifications may result from machine learning techniques (e.g., Random-Forest Classifiers) or may be derived from simple if-then rule-based methods. They sometimes also provide diagnosis suggestions telling what caused the fault. The key aspect of these methods is that they provide a clear assessment of whether measurements are anomalous or not. Hence, the user can directly focus on diagnosis or evaluating the correctness of the results.

Prediction. In analogy to the quantitative model-based methods from Isermann and Ballé, this strategy provides predictions that can be compared to measured data. However, no distinction is made whether these results come from quantitative-model-based (e.g., Parity Space methods) or process-history-based algorithms (e.g., Random-Forest Regression). Instead, the key aspect of this strategy is that predictions for measurements are available, which can be compared to measured data. Thus, this provides additional information about how much the system deviates from its expected behavior. In addition, an assessment of whether measurements are anomalous or not is often provided as well.

4 Data and Task Abstraction

This section describes the data used for Fault Detection and Diagnosis at solar-thermal systems and the abstract tasks of the target users. This information was mainly gathered by working closely with two domain experts from SOLID in two contextual inquiry sessions and two monitoring training sessions (see Methodology in section 2).

4.1 Data Characterization

The result of the data characterization can be seen in Figure 6. It shows the datasets that are used for Fault Detection and Diagnosis at solar thermal plants. The 13 datasets have been grouped into three categories based on their source: *Sensor Data*, *Event Data,* and *Anomaly Algorithm Results*, which are described individually in further detail below.

		Dataset	Dataset Type	Attributes
Sensor	S1	Details	table	sensor name, unit of the measurements
	S2	Measurements	time-series	timestamp, measurement value
	S3	Position	network	positions of sensor (either topological or in terms of connected parts of the system)
	S4	Correlations	network	links between correlated sensors
Event	E1	Details	table	type, status, severity, title, description of the event
	E2	Intervals	table	start, end of event
	E3	Affected sensor	lists	list of sensors that are affected by event
	E4	Relations	network	links between events, type of relation
Anomaly Algorithm	A1	Details	table	ID, class of algorithm, description
	A2	Target	table	target sensor
	A3	Results	table	start, end, probability of fault
	A4	Predictions	time-series	timestamp, predicted sensor values
	A5	Input Features	network	directed link from input to target, weight of correlation

Figure 6: Summary of data characterization results, listing the needed datasets for FDD, their types, and their most important attributes. All 13 datasets can be categorized into three sections: sensor data containing information about sensors and their measurements; event data containing information about events influencing the system, like services or faults; and anomaly data containing information and results of fault-detection algorithms. The terminology of Munzner [16] was used to determine the dataset type.

Sensor Data. The basis for analyzing solar thermal systems is sensor data. Based on guidelines for monitoring solar thermal plants [24, 25], a minimum of around 20 sensors are needed per system to keep track of essential measurements. However, real applications typically have hundreds of sensors installed, depending on the size of the plant. Measurements (S2) are stored as timestamp value pairs for each sensor in approximately 1-minute to 5-minute intervals. In addition, monitoring personnel also need semantic information (S1) about the sensors and components. The most important is the unit of measurement and the sensor name, but additional details like component configurations, sensor accuracy, installation date, or similar data are also sometimes needed. Especially the position of the sensor (S3) and the derived hierarchy and plant layout is essential for monitoring, as it allows the monitoring personnel to understand the flow of the fluids. Without this information, they cannot interpret the sensor values. Thus, displaying the system layout is central in most SCADA (supervisory control and data acquisition) systems and other industry-related visualization approaches (see related work in section 5). In addition, there are also logical correlations (S4) to keep in mind, which are based on the physical relationships or system-control setpoints. For example, the solar flow temperatures might be controlled based on the current demand, forming a logical correlation between demand and flow temperature. Unfortunately, system-control information is often hidden in technical description sheets and is rarely available.

Event Data. Another piece of information needed by the monitoring personnel is contextual information about events that influence the system's performance. For example, tests conducted during services can explain abnormal sensor measurements, and known faults might help diagnose new anomalies. In this work, *events* are considered intervals with a start and end time, in contrast to the Maintenance Terminology Norm *BS EN-13306:2017* [21], which considers events instantaneous. This usage is backed up by software like smartblue [26], for example, which also uses the word *event* to describe periods. In addition to the start- and end-time of the events (E2), the monitoring personnel needs detailed context information about the event (E1). These may be stored using summative descriptions but could also contain comments or attachments to allow flexible and comprehensive documentation. Another piece of information that is needed is the affected parts and sensors (E3). For example, events can affect the whole system or only a small subsection, such as a specific component or sensor. Finally, events might have relations to other events (E4). For example, a fault may be the cause of another fault. These relations are needed to understand the evolution of a fault.
For this work, *events* have been separated into three types based on their meaning to the domain user – *faults*, *activities,* and *info*. First, *faults* contain information about unpermitted deviations in the system behavior (see section Terminology). Deriving them is the primary goal of the monitoring personnel. For completing diagnosis, the current status (e.g., active, latent, ended) and the fault's severity often need to be estimated. This information allows system managers to prioritize and plan services and repairs. Second, *activities* like services or repairs can be defined as permitted interventions in plant operation. While their effects on the system are typically positive, these events might also explain unusual data, for example, if tests are conducted, or parts of the system are changed. Finally, *infos* are additional context information to exchange notes between analysts. For example, this can be used to document individual system specifics, like less energy demand on weekends.

***Anomaly Algorithm Results*.** Finally, the data generated by different anomaly detection algorithms must be used to benefit from the automatization. The main contributions of all anomaly algorithms are their detected anomalies (A3), which share similar characteristics to the event data, namely, start and end time. In addition, probability scores may be added to let the user gain more insights into the accuracy of an algorithm result. All anomaly algorithms are typically connected to a target sensor or part (A2) and may contain additional descriptions of how the algorithm works (A1). Some quantitative model-based and history-based anomaly algorithms also provide prediction values (A4) which can be compared to the measurements. Another information that can be used is the knowledge about which input feature and target-feature combinations (A5) are used by the algorithms, as this gives insights about correlated measurements. A challenge is that many anomalies might be detected depending on the sensitivity and number of algorithms. Assuming hundreds of sensors and tenths of algorithms per sensor, it is impossible to study individual algorithm results in detail.

4.2 Task Abstraction

Based on the interviews and inquiry sessions with the domain users, the following high-level tasks have been identified:

T1. Overview. This task serves as a starting point for the fault detection and diagnosis workflow, giving the monitoring personnel a “big picture” of the system and allowing them to prioritize their work. An overview is needed on two fronts: First, the system status needs to be assessed by checking for recent events and detected anomalies. Thus, one can detect patterns or trends to identify problems that need to be investigated in further detail. Second, an overview of the

system layout is needed to understand the sensor correlations better. Doing this early allows the user to carry out subsequent tasks more efficiently.

T2. Evaluate Anomalies. If anomalies are found, they need to be investigated. In the end, the monitoring personnel needs to judge if detected anomalies result from faults, bad algorithm results, or just a product of rare operating conditions. Thus, solving this task includes exploring correlated sensors and related events and checking the sensor data for further clues.

T3. Manual Fault Detection. As automatic algorithms might not identify all faults, monitoring personnel must manually check the system for additional unidentified faults.

T4. Fault Diagnosis. When a fault has been identified during tasks T2 or T3, additional information about its cause and consequences must be derived. Thus, their task is identifying affected sensors and the fault's location, time, root cause, and severity**.**

T5. Annotation of Events. Finally, information about events needs to be annotated to benefit from the derived knowledge. This annotation allows sharing the information with fellow analysts and system managers. In addition, the information about known events might be helpful in future monitoring sessions to explain anomalies, diagnose similar faults, or find root causes.

4.3 Design Goals.

Additionally, the following requirements should be satisfied:

G1. Efficient workflow. The focus is on efficiently directing the user to potential faults at the expense of undirected exploration. Otherwise, the detection and diagnosis of faults will take too long and monitoring many systems on a regular basis will not be feasible anymore.

G2. In-depth analysis. If a user wants to investigate a fault in a high level of detail, this must be possible without switching to another application.

G3. High coverage of faults. The number of undetected faults must be as small as possible, and all critical faults must be detectable.

G4. Exportable Results. It must be possible to store and export information about detected faults and other events. As described in Task *T5 Annotation of Events*, this allows the users to better communicate and document events.

G5. Scalable to other solar thermal systems. The application should work for any arbitrary solar thermal plant, independent of its size and layout. The application should thus support up to multiple hundreds of sensors per system.

4.4 Relations between Tasks and Data

For further analysis, *Figure 7* shows the abstract tasks from section 4.2 in relation to the needed datasets from section 4.1. In addition, questions that need to be solved to complete the tasks are shown. Data considered essential for solving a question is marked in darker colors, whereas optional helpful data is displayed as less saturated. The information to draw this plot was gathered during discussions and sessions with the domain user and through feedback to rapid prototypes. However, distinctions between “essential” and “helpful” are rather vague and are highly influenced by the author's personal opinion. However, some basic patterns can be deduced, nevertheless. This analysis might indicate which part of the data is needed at which step of the process. It allows us to better understand the importance of each dataset and how they are used.

T1 Overview. Questions Q1-Q3 mainly deal with gaining an overview of the system status. Thus, the most relevant details of recent events and anomalies need to be summarized. In contrast, the overview of the system layout (Q4) can be shown by primarily relying on the sensor position data. Because task T1 serves as an overview, it thus contains summarized data

from all three dataset categories.

T2 Evaluate Anomalies. In contrast, task T2 focuses on the sensor data and anomalies, hardly involving any event data. Questions Q5-Q9 relate to understanding why an anomaly algorithm alarm has been triggered. Thus, measurement values and the time of the anomaly are essential for solving these questions. In addition, algorithm inputs and outputs may get investigated to make sense of why the algorithm raised the alarm. In contrast, questions Q10-Q11 are less concerned about sensor data but instead try to assess the quality and trustworthiness of the algorithms.

T3 Fault Detection. Dealing with manual fault detection, question Q12 mainly requires the sensor data. Here, monitoring personnel might look for any patterns that indicate bad system behavior. Alternatively, critical measurement quantities might be checked specifically to identify faults. Events may also trigger further investigations, for example, if monitoring personnel expect other sensors to be affected. However, manual fault detection does not involve automatic anomaly detection results at all.

T4 Fault Diagnosis. Almost all of the identified data categories play a role in fault diagnosis. Here, the task of the monitoring personnel is to gather information from multiple angles of perspectives, describe the fault more clearly, and highlight relations. For these tasks, the distinction between essential- and helpful data is more difficult as well.

T5 Annotation. Finally, all the information from sensor values and anomalies must be combined and documented to make them available for reporting and further analysis sessions.

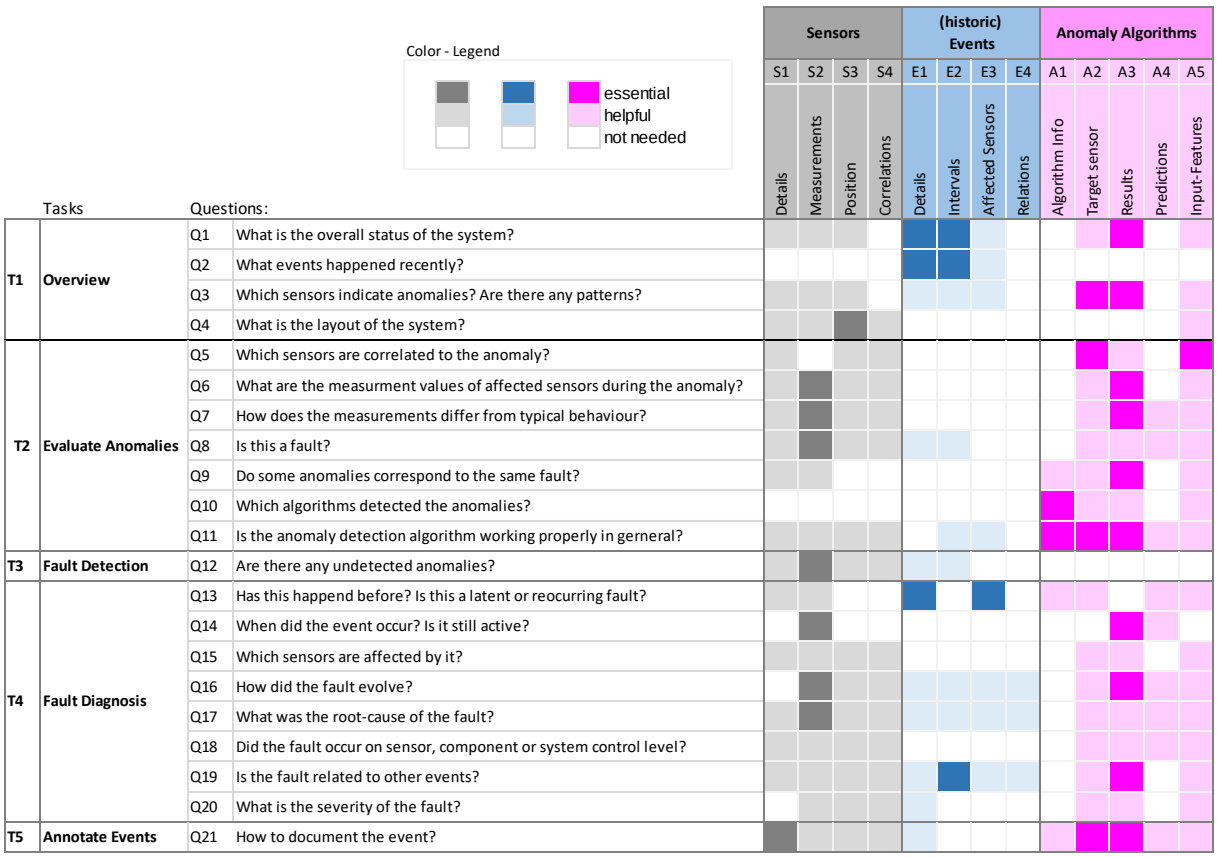


Figure 7: Table showing high-level tasks and domain user questions in relation to the needed monitoring data identified in the data abstraction. Data considered essential to solving a question is marked by dark-colored boxes, while optional supporting data is marked in lighter colors.

5 Related Work

This section shows related work to Fault Detection and Diagnosis. First, contributions from the solar-thermal domain are presented before discussing existing monitoring applications for the quite related photovoltaic domain. Finally, the last part of this section deals with VIS design studies concerning Fault-Detection. An overview of the results of this section can be seen in Figure 8 and Figure 9, showing the relation of each contribution to the identified abstract tasks and data from section 4.

5.1 Solar-Thermal Domain

In the solar thermal domain, FDD is mainly researched as part of Monitoring or developing new Fault-Detection algorithms. For example, Knabl et al. [25] and Feierl and Luidolt [24] published monitoring guidelines providing information about gathering-, storing-, and analyzing sensor data. Alternatively, Kainzer et al. [27] and Faure et al. [28] provide an overview of published fault detection methods for Solar Thermal Systems. They discuss each method's benefits and shortcomings but conclude that few commercial applications for FDD exist yet [28], and more research needs to be done. However, they do not include any information regarding visualization.

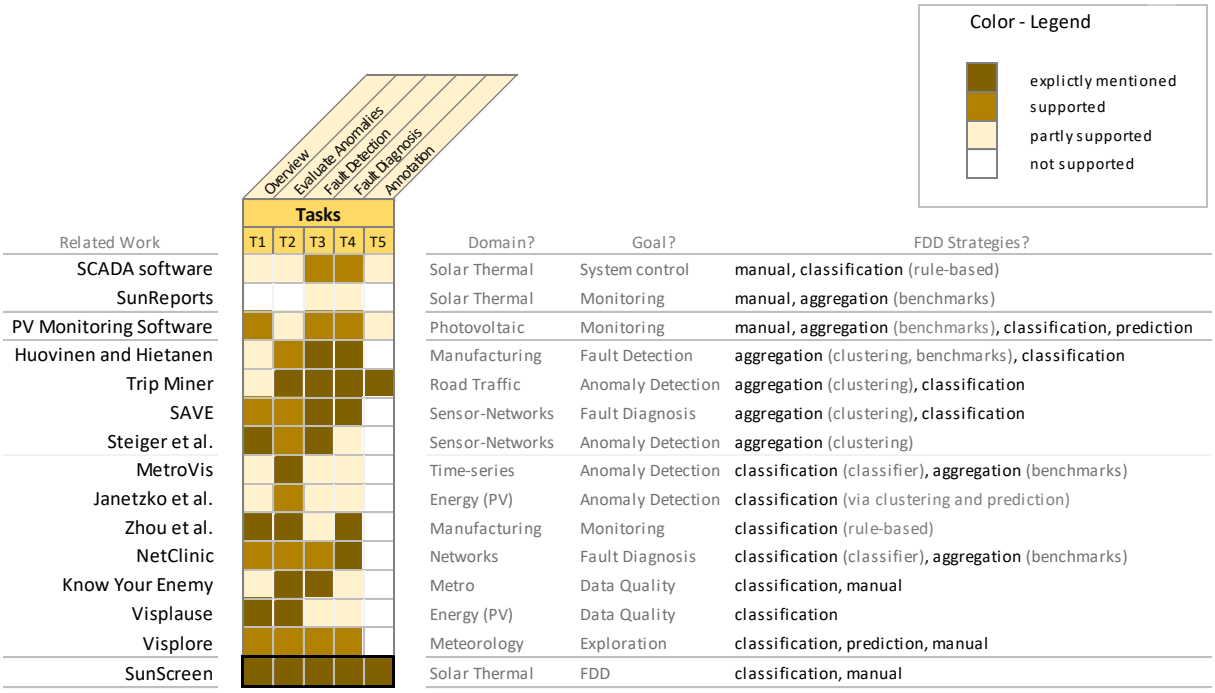
One of the rare exceptions of visualization software for fault detection for the solar thermal domain is *SunReports* [29], which provides a basic line chart of some sensor values and benchmarks for different time intervals. However, they do not use automated fault detection, and their product is not scalable to hundreds of sensors.

Missing other options, monitoring personnel typically rely on system control and data acquisition (SCADA) software [1, 2, 3]. These applications allow them to check the sensor data and control the system remotely. They feature a topological view of the sensor positions, their current measurements, and line charts of selected sensor measurements. Some software like Atvise [3] even provides functionalities to set custom alarms and annotations as well. However, as these systems are not directly targeted at Fault Detection of solar thermal systems, they lack the support for analyzing multi-variate time-series data and integrating more complex anomaly detection algorithms.

5.2 Photovoltaic Domain

While not much commercial software can be found for solar-thermal applications, many more fault-detection applications are available for photovoltaic (PV) solar systems [30, 31, 32, 33, 34, 35, 36, 26]. Most of them feature a system portfolio as an overview, showing a list of all systems, including their current power generation, severity status, open tickets, and other benchmarks. The recent power generation is often displayed as a spike-line for each system in the list. Automatic fault detection is done by checking for basic faults like missing data or comparing measured- with expected power generation. The latter is either estimated via machine learning algorithms [36, 31, 34, 35, 26], via simulation models based on irradiation [36, 30, 32, 34, 35, 26], or via comparisons with other systems [36, 30, 31, 33, 34, 35, 26] using either heatmaps or line charts for displaying the results. If more detail is needed, the sensor data can be shown via line charts, with colored backgrounds for periods where faults occurred, sometimes showing a summary description.

While most of these applications cover many identified tasks and data, they still cannot be used for the solar-thermal domain. One reason is that the underlying data of solar-thermal applications seem to be more complex than in the case of PV. This complexity may be explained by the lower inertia of the solar-thermal fluid and the higher complexity of many factors (like irradiation, consumer demand, and storage capacity) mutually influencing the system's temperatures. As such, much more measurands must be shown simultaneously to understand the data, and the sensors' position seems much more relevant, too. In addition, the automatic algorithms would need to be adjusted for the solar thermal domain.



Related Work	Domain?	Goal?	FDD Strategies?
SCADA software	Solar Thermal	System control	manual, classification (rule-based)
SunReports	Solar Thermal	Monitoring	manual, aggregation (benchmarks)
PV Monitoring Software	Photovoltaic	Monitoring	manual, aggregation (benchmarks), classification, prediction
Huovinen and Hietanen	Manufacturing	Fault Detection	aggregation (clustering, benchmarks), classification
Trip Miner	Road Traffic	Anomaly Detection	aggregation (clustering), classification
SAVE	Sensor-Networks	Fault Diagnosis	aggregation (clustering), classification
Steiger et al.	Sensor-Networks	Anomaly Detection	aggregation (clustering)
MetroVis	Time-series	Anomaly Detection	classification (classifier), aggregation (benchmarks)
Janetzko et al.	Energy (PV)	Anomaly Detection	classification (via clustering and prediction)
Zhou et al.	Manufacturing	Monitoring	classification (rule-based)
NetClinic	Networks	Fault Diagnosis	classification (classifier), aggregation (benchmarks)
Know Your Enemy	Metro	Data Quality	classification, manual
Visplause	Energy (PV)	Data Quality	classification
Visplore	Meteorology	Exploration	classification, prediction, manual
SunScreen	Solar Thermal	FDD	classification, manual

Figure 8: Correlation between related work and identified tasks shown as a heatmap. Related work explicitly mentioning the tasks inside their requirement analysis (or similar chapters) is highlighted in a dark color. In contrast, lighter colors mean that the task is at least supported or partially supported by the application. In case a task is not supported, the cells are colored white. In addition, the right side of the table shows the domain and the goal of each related work, as well as which strategy was used to help the user perform fault detection and diagnosis (see Terminology in section 3.3).

Related Work	how?
SCADA software	linechart (S1, S2) w. coloured background(E2, A3), sensor topology (S3). list (E1-E3), list (A2, A3, A5)
SunReports	linechart (S1, S2), aggregation to benchmarks (S2)
PV Monitoring Software	linechart (S1, S2) w. coloured background (E2), sensor positions (S3), heatmaps (S2, A3). grouped lists (E1-3, A1-A4)
Huovinen and Hietanen	linecharts (S1, S2), conditional histograms (S2, A2, A3) in subplots for each indicator.
Trip Miner	geoposition map (S3, S2), circular dimension projection (S2, A3), linechart(S2, A3), heatmatp (S2, A3), histogram (S2)
SAVE	topology graph (S3, S2, A3), correlation graph (S1, S4, A3), radial dimension projection (S1, S2), linecharts (S1, S2)
Steiger et al.	topology graph (S3, S1) with calendar plot (S2) and similarity graph (S2), similarity graph of daily patterns (S2)
MetroVis	linechart (S1, S2) w. coloured background (A3), barchart of statistics (A3) details of anomalies (A1, A3), Venn-Diagram(A3)
Janetzko et al.	treemap (S3) with linecharts (S2) w. colored background (A2,A3) or spiral charts or calendar view (S2, A2, A3)
Zhou et al.	workshop view and subsection view (S3) w. components coloured by status (A2, A3), linechart (S2) w. horizon graph (A3)
NetClinic	network graph with multiple levels (S2, S3, S4) and highlighted anomalies (S4, A1-A3, A5), list (A1), list with histogram (A1, A3)
Know Your Enemy	table (S2), heatmap (S2, A3), statistics (S2, A3)
Visplause	linechart (S1, S2) w. coloured marks (A3), heatmap (S2, S1, A1, A2, A3), table (S2)
Visplore	linecharts (S1, S2) w. coloured marks (A3), heatmap (S2, S1,A2,A2, A3), many more (S1, S2, S4, A1-A5)
SunScreen	linecharts (S1, S2), heatmap (A2, A3), gantt-chart (E2), list (E1-E3), table (A1-A4, A5)

Figure 9: Correlation between related work and data shown as a heatmap. Related work that uses specific data properties is colored in darker colors. The cells are colored in lighter colors if properties are only partially used, while white indicates that the data is not used at all. In addition, the right side of the table shows how each related work displays the data.

5.3 Visualization Design Studies

Applications to support fault detection have been described in various other domains too. Multiple design studies have been conducted focusing on this topic. The related work below is grouped based on their FDD strategy (see FDD Strategies in VIS in section 3.3).

5.3.1 Aggregation-based strategies

Several works have been proposed that use aggregation-based strategies. They use techniques (e.g., clustering) to group the data based on its properties. Anomalies are then found by checking for outliers either manually or automatically:

> **Automatic.** Two examples where anomalies are detected automatically are Huovinen et al. [23] and Riverio et al. [7]. Huovinen et al. [23] work with data from a pulp dry facility. They assume that a large portion of their data corresponds to "normal" operating points and try to find these modes using clustering. Outliers that do not fit into these clusters are then regarded as anomalies. TripMiner [7] also uses the same approach to detect accidents or near accidents in road traffic data. Again, clustering is used to find normal driving behavior, and outliers are labeled as anomalies. Unfortunately, such an approach is expected to fail in the case of solar-thermal data. That is because temperatures, volume flow, irradiation, and demand can change both gradually and drastically. This way, there is no clear "steady-state" operating point, and thus no confined cluster will be found.
>
> **Manually.** Shi et al. [5] and Steiger et al. [6] also use clustering to group the data. However, the authors do not make any assumption about whether a group corresponds to typical behavior. Instead, they view the data from multiple perspectives, letting the user decide whether it is anomalous. For instance, SAVE from Shi et al. [5] focuses on sensor networks and supports detecting routing problems between sensors. To do so, they visualize the routing paths, the correlations between sensor measurements and status, and the temporal evolution of routing paths and measurements. The user can then check the graphs for patterns and correlations.
>
> Another example is Steiger et al. [6], which try to detect anomalies of power consumption in an energy grid. They use clustering to group similar daily patterns of each of the sensors together. To explore the results, their visual framework shows the clusters in a scatterplot, positioning similar patterns next to each other. One limitation is that they demand univariate data and that sensors measure the same quantity only. Another drawback for both discussed frameworks is that the anomaly detection is not automated, such that users need to explore the data themselves. Thus, this might conflict with goal G1 Efficient Workflow.

5.3.2 Classification-based strategies

Other related work mainly relies on classification-based fault detection strategies. These systems use domain-knowledge-based rules or machine-learning approaches to identify anomalies automatically. Thus, labels (i.e., anomaly or no-anomaly) are assigned to all datapoints. The classification results are then displayed in various representations, depending on the target domain and data. Related work below is grouped based on their goal to provide better structure for the reader.

> **Detection.** Some authors explicitly focus on the task of Anomaly Detection (T2). Unfortunately, often less attention is paid to other tasks like Fault Diagnosis. Thus, the visual representations of these works often provide less flexibility in exploring and analyzing the data: For instance, MetroVis from Eichmann et al. [12] supports data scientists in evaluating their FDD algorithms. The results are displayed using line charts, and some functionality is provided to compare different algorithm results. However, MetroVis only supports showing one sensor at a time, which is insufficient to understand the complex behavior of solar-thermal systems

or diagnose anomalies.
As another example, Janetzko et al. [13] focus on anomaly detection for power consumption data. A treemap is used to display the hierarchy of the sensors. The sensor data is displayed in each panel using spiral graphs, calendar views, or line charts, while anomalies are displayed using color encoding. However, the treemap layout does not scale to the hundreds of sensors used at solar thermal systems.

Diagnosis. In contrast, work focusing on fault diagnosis does support almost all of the identified tasks. Unfortunately, none of the proposed frameworks supports time-dependent multivariate data of solar thermal systems:
For example, Zhou et al. [11] focus on monitoring a large manufacturing facility. Using a rule-based method, a criticality index (i.e., anomaly score) is assigned to each part of the system for its current-, short-term-, and long-term behavior. The data is displayed showing the system's layout, with components colored based on their anomaly score. The user thus gets an intuitive overview (T1) while anomalies can be evaluated (T2) and further diagnosed (T3) in subsequent views - showing subparts of the system, anomalies, and sensor values. Here, the authors chose a color encoding to visualize the temperature variations compared to its reference for each zone. However, this requires that constant reference temperatures are available. Unfortunately, such an encoding is impossible for solar thermal data, as temperatures fluctuate depending on radiation and demand. In addition, details of only one zone can be shown at a time, while analysis of solar thermal applications requires information on multiple parts of the system simultaneously.
Another inspiring example is NetClinic from Lui et al. [4]. They aim to support fault diagnosis for computer networks, using the results from an external diagnosis tool. Conveniently, the tool already provides suggestions for potential culprits as well (T4). The data is displayed using a network graph, showing machines and their components in circular hierarchical nodes and their correlations as links between nodes. Another view contains the list of anomalies, highlighting the path from the culprit to the affected machine on click. The user can thus either explore the detected anomalies (T2) or search for anomalies and culprits themselves (T3). Unfortunately, the visual framework cannot explore the temporal evolution of the data. While this is not critical to diagnose faults in computer networks (e.g., wrong router/firewall configurations), solar thermal data cannot be understood without a temporal context. In addition, no comparable sensemaking algorithm that provides diagnosis suggestions is available for solar thermal applications yet.

Data-Quality. Finally, some approaches focus on data quality assessment. Here, the authors use anomaly detection to find missing data or other quality problems. Typically, this is done to prepare the data for further data processing steps or to infer improvement suggestions:
One example is KnowYourEnemy from Gschwandtner et al. [10]. Their primary goal is to check the data for quality issues. Users can either rely on automatic checks (T2) or investigate the data themselves (T3) to find anomalies. Results are displayed using a heatmap, which can be configured to show combinations of measurements, anomaly scores, and time in different granularities. However, the heatmap can only show a maximum of three sensors at a time, while many more correlations might be needed to understand specific behaviors of solar-thermal systems.
Visplause [8] focuses on data quality assessment, too. Like KnowYourEnemy, they let users explore the automatic anomaly detection results. An overview of anomalies versus time is provided using a heatmap. Each lane can be extended to show sensors and algorithm types while time is shown on the x-axis. The size of the cells depicts the number of anomalies found for each period, and the color depicts the type of anomaly algorithm which found the issue.

Measurement values of selected sensors are displayed redundantly using a table and a line chart. This visual framework is considered the most similar to *SunScreen*. However, some tasks like annotating faults (T5) or manual fault detection (T3) are not directly supported. As a result, no contextual information can be accessed during diagnosis, and exploring results seem to focus on one sensor at a time only.

Some of these issues were resolved by the more advanced Visplore [9], developed by the same authors. For example, checking the details of multi-variate data is possible by displaying multiple sensor measurements in subplots next to each other. Very recently, an annotation functionality was added to the system as well. In addition, many other functionalities like correlation analysis and different data representations further allow exploring the data to a greater extent. However, the extensive functionality conflicts with goal *G1 Efficient workflow,* as users might get distracted or overwhelmed by the sheer number of potential choices for displaying the data.

6 SunScreen

This section presents the visual framework *SunScreen* that was developed using the results from the requirement analysis described in sections 4 and 5. This section first describes some implementation details and gives a brief overview of *SunScreen*. Finally, design decisions are justified in the last part of this section.

6.1 Implementation Details

SunScreen is implemented in vue.js, primarily relying on the libraries D3 [37] and Plotly [38]. Real-life data is provided by SOLID and includes almost all data identified in the data abstraction (see section 4.1. The abbreviations from Figure 6 are used in the text below for better comparison). The data is fetched from three MySQL databases using HTTP requests. One of the databases stores events (E1-E4) that were manually inserted by the monitoring personnel to annotate services or faults. Another database is used to store the sensor data (S1-S4) from multiple systems, with about 200 sensors per system and data available since 2016. However, relations between events (E5) and sensor correlations (S5) are unavailable. Finally, an anomaly database provides results from automatic fault detection algorithms (A1-A5). More specifically, the following types of algorithms are supported:

- **Missing data.** Checks if any sensor data is missing or contains NAN-values (i.e., Not A Number).
- **Constant data**. Checks if sensor values stay constant for too long.
- **Collector stagnation.** Checks whether the collector temperature exceeds 130°C.
- **SunSearcher.**[1] Machine learning algorithm that tries to detect correlations between any sensor measurements. If sufficient correlations are found, a random forest regressor is trained and used to provide predictions for the target sensor. In case the offset between predicted and measured values is too high, an anomaly is raised.
- **Collector Similarity.** Simple linear regression that tries to predict the measurement values of a collector temperature sensor. It uses the sensor values from other collector temperature sensors as input features. However, this algorithm was only used until the usability test because it created too many false positives. Hence, this algorithm class and the corresponding anomalies were excluded from *SunScreen* after the usability test.

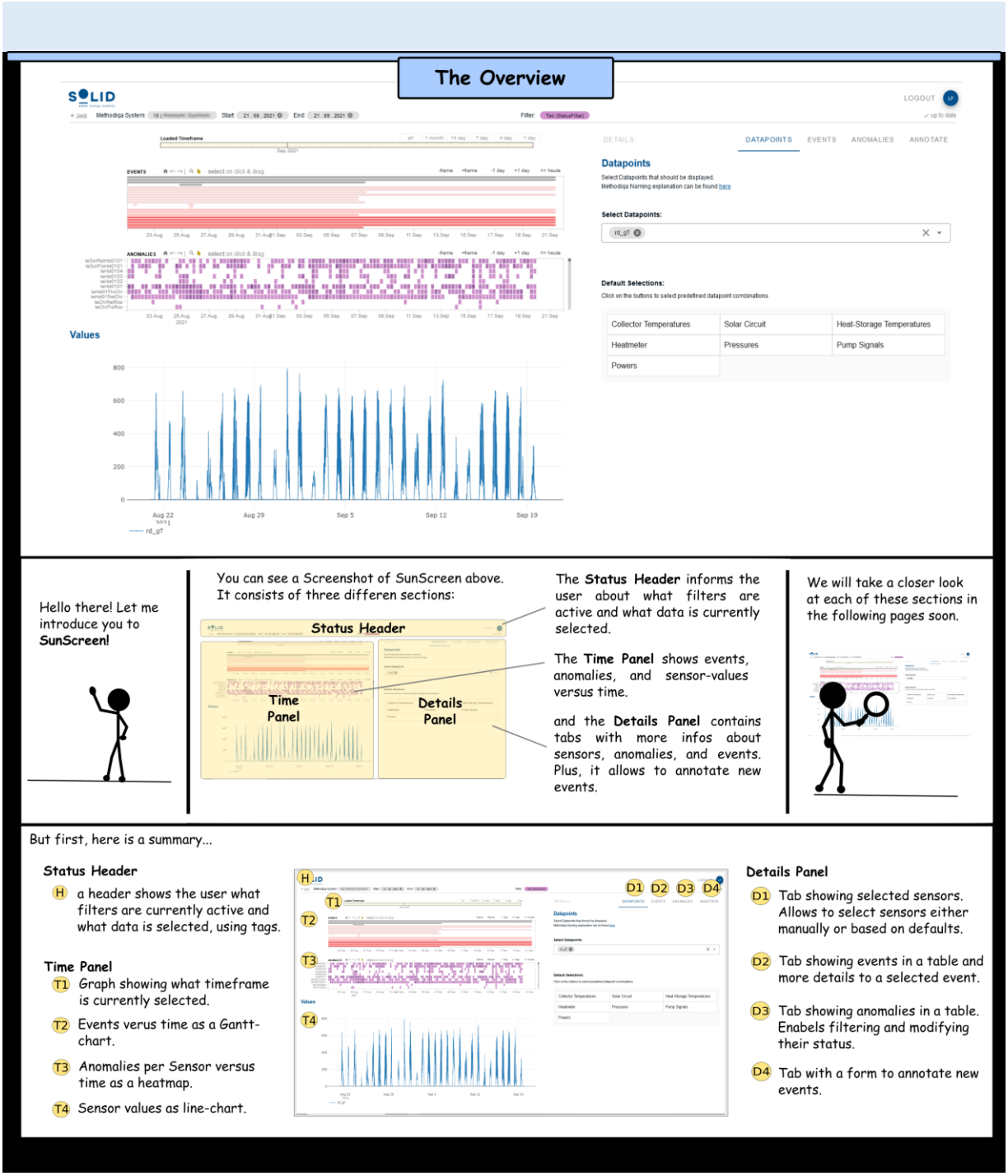
For each of the algorithm types above, many individual algorithms might be created, targeting different sensors. For instance, algorithms from the *collector stagnation* class are assigned to each

[1] This algorithm was inspired during an EU-funded project called Ship2Fair. The details to the algorithm will be published in a separate paper, which is currently in review under the name of *FaultDetective*.

collector temperature in the dataset. Each algorithm then might detect anomalies for the corresponding target sensor.

6.2 Overview

An overview of *SunScreen* is provided in the panel below, using a data comic [14]. The idea behind the comic-like representation is to make it easier to understand interactions and to limit switches between text and figures (see section 8.3 for a discussion on the benefits of data comics).
The comic first provides a rough overview of *SunScreen* before dealing with each view individually.



Status Header

Header

The status header displays information about which filter are applied and what is currently selected.

Each piece of information is displayed as a tag.

tagname value ×

The values can be altered by clicking on the tag and changing the value.

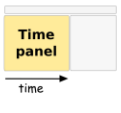
start 21. .2021 ×

Or reset the tag by clicking on the x

start 01.08.2021 ×

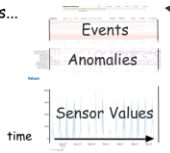
* system name has been blurred for confidentiality reasons.

Time Panel



The **Time Panel** shows the data versus time.

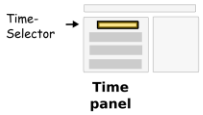
It displays...



... using a shared x-axis

On the top, a small graph shows the user what timeframe is currently accessible and what time-period is shown in the views below the graph (zoomed).

-> The Time-Selector



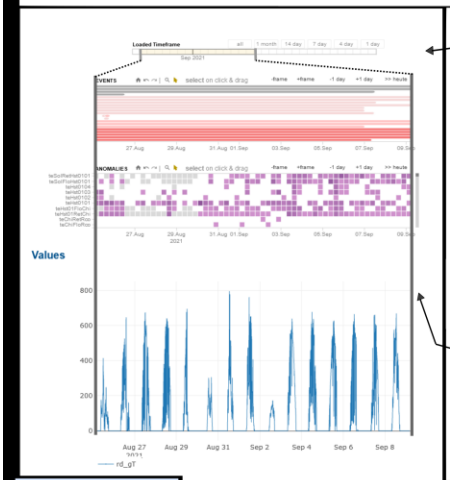
The monitoring personnel is primarily interested in **recent** system behavior.

Thus, by default only 1 month is loaded and displayed

But often, data must be checked in more detail.

5min details

The Time-Selector is supposed to give the user more context, what timeframe he is currently looking at..

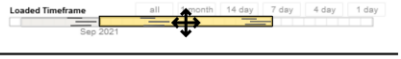


The yellow colored region shows which timeframe is selected in the views below

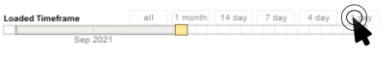
It can be altered ...

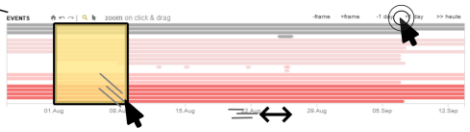
... selecting a new timeframe by brushing

... shifting the timeframe by dragging



... setting the width of the timeframe by button click.





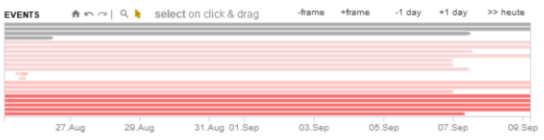
Similarly adjusting the timeframe is also possible in each of the views below, ...

... using the the zoom tool
... by dragging the x-axis
... using the buttons on the top

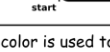
-> The Event-Gantt

The events are shown directly below the Time-Selector

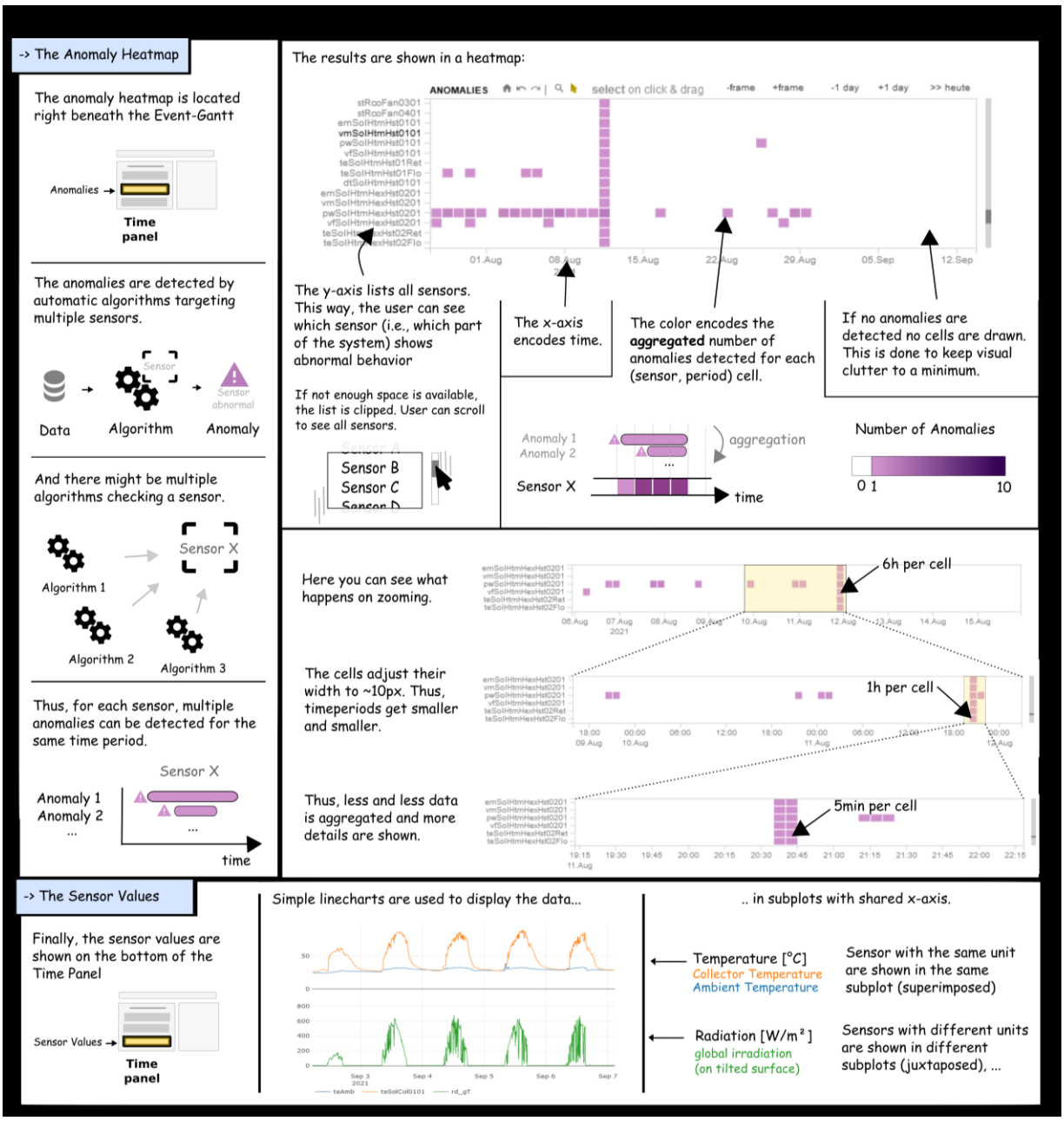
Events are displayed using a Gantt-Chart.



encoding the start and end time of the events by the position- and length channel



while color is used to encode the type of the event


-> The Anomaly Heatmap
The anomaly heatmap is located right beneath the Event-Gantt
Anomalies
Time panel
The anomalies are detected by automatic algorithms targeting multiple sensors.
Sensor
Sensor abnormal
Data
Algorithm
Anomaly
And there might be multiple algorithms checking a sensor.
Sensor X
Algorithm 1
Algorithm 2
Algorithm 3
Thus, for each sensor, multiple anomalies can be detected for the same time period.
Sensor X
Anomaly 1
Anomaly 2
...
time
The results are shown in a heatmap:
ANOMALIES
select on click & drag
-frame
+frame
-1 day
+1 day
>> heute
stRooFan0301
stRooFan0401
emSolHtmHst0101
vmSolHtmHst0101
pwSolHtmHst0101
vfSolHtmHst0101
teSolHtmHst01Ret
teSolHtmHst01Flo
dtSolHtmHst0101
emSolHtmHexHst0201
vmSolHtmHexHst0201
pwSolHtmHexHst0201
vfSolHtmHexHst0201
teSolHtmHexHst02Ret
teSolHtmHexHst02Flo
01.Aug
08.Aug
15.Aug
22.Aug
29.Aug
05.Sep
12.Sep
The y-axis lists all sensors. This way, the user can see which sensor (i.e., which part of the system) shows abnormal behavior
The x-axis encodes time.
The color encodes the aggregated number of anomalies detected for each (sensor, period) cell.
If no anomalies are detected no cells are drawn. This is done to keep visual clutter to a minimum.
If not enough space is available, the list is clipped. User can scroll to see all sensors.
Sensor B
Sensor C
Anomaly 1
Anomaly 2
aggregation
Sensor X
time
Number of Anomalies
0 1
10
Here you can see what happens on zooming.
6h per cell
The cells adjust their width to ~10px. Thus, timeperiods get smaller and smaller.
1h per cell
Thus, less and less data is aggregated and more details are shown.
5min per cell
-> The Sensor Values
Finally, the sensor values are shown on the bottom of the Time Panel
Sensor Values
Time panel
Simple linecharts are used to display the data...
.. in subplots with shared x-axis.
Temperature [°C]
Collector Temperature
Ambient Temperature
Sensor with the same unit are shown in the same subplot (superimposed)
Radiation [W/m²]
global irradiation
(on tilted surface)
Sensors with different units are shown in different subplots (juxtaposed), ...

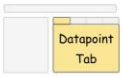
Details Panel

The **Details Panel** can be used to analyse the data on the Time-Panel in more detail.

It contains four different tabs:

-> The Datapoint Tab

The first tab focuses on the datapoints (a.k.a. sensors)



It allows user to manually select sensors that are then shown in the Time-Panel on the left

Thus, it serves as a starting point for the Manual Fault Detection task (T3)

Selected sensors are shown as tags. They can be deselected by clicking on the ˣ.

Sensors can be selected manually, using a auto-completion menu.

This section contains default selections. Frequently used sensors can be selected quickly, by using the toggle switches.

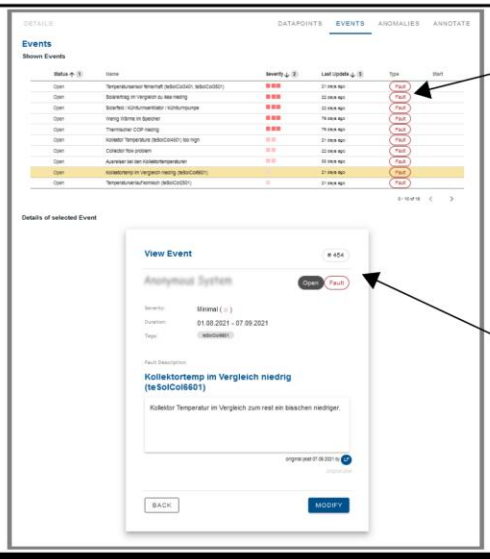
Selections are grouped based on the section of the system they belong to. For example, **Sol** contains default selections for the solar primary circuit.

-> The Event Tab

The second tab deals with the event-data.

It allows the user to get more information about the displayed events in the Event-Gantt

Thus this information is relevant for getting more details during the Overview (T1), Evaluation (T2) and Diagnosis (T4) tasks.



The top shows a simple table with all events, that are currently shown in the Event-Gantt chart.

The table includes:

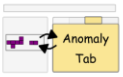
Open	the status
Wenig Wärme im Speicher	the title
■■■	the severity
21 days ago	time since the last update
Fault	the type (fault/activity/info)
15.04.2021	the start date

The bottom shows one selected event in more detail. It also contains a description and who annotated the event

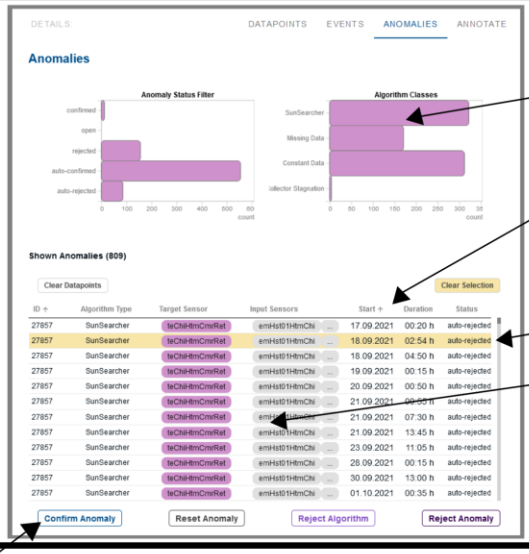
-> The Anomaly Tab

The next tab displays more details about the anomalies.

While the Time Panel only shows aggregated anomaly data ....



... the Anomaly Tab, in contrast, allows to interact with individual anomalies and to display all correlated information.



The histograms on top allow to filter the anomalies based on their status (see panel below) and based on the type of algorithm.

The table shows all anomalies that are shown in the Anomaly Heatmap.

It displays additional information like the used input-sensors, duration and start of the anomaly and some infos about the used algorithm.

on click on a row, the corresponding anomaly is selected.

On click on a sensor-tag, the data is displayed in the SensorValues view in the Time-Panel.

Tags are hidden if not enough space is available. In this case, the sensor names can be shown by clicking on the ...

In addition, the status of the selected anomalies can be altered using the Buttons on the bottom of the table.

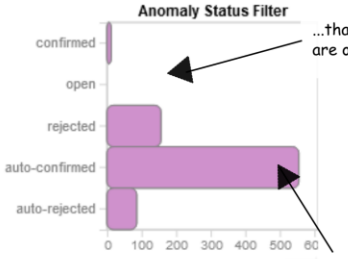
The effects of each button click can be seen on the right ->

Button?	When to use?	What happens?
Confirm Anomaly	the anomaly was detected correctly (true-positive)	anomaly => *confirmed*; New Event annotated; related anomalies* => *auto-confirmed*
Reset Anomaly	the anomaly should be set to *open* again	anomaly => *open*
Reject Algorithm	the algorithm seems to yield only bad results. (many false-positives)	algorithm => *disabled***; related anomalies*** => *auto-rejected*
Reject Anomaly	the anomaly does not correspond to a problem (false-positive)	anomaly => *rejected*

* all anomalies during the time of the event, which use any of the affected sensors.
** algorithm is disabled permanently, and will not raise alarms anymore.
*** all anomalies that were detected by the rejected algorithm.

The histograms on the top displays the number of anomalies that are currently shown for each category.

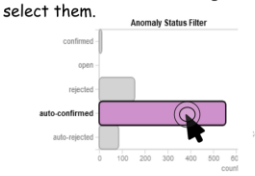
For instance, this shows...



...that no anomalies are open anymore...

..and most of the anomalies have already been (auto-) confirmed.

A user can click on the categories to select them.

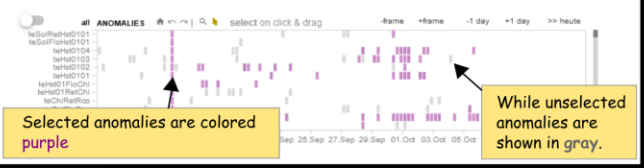


Unselected categories are then filtered out in the other views...

In the AnomalyTable, only data of selected anomalies are shown. All other rows are filtered out.

only shows "auto-confirmed" anomalies

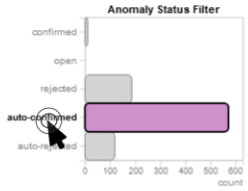
In the AnomalyHeatmap, the color is modified accordingly.



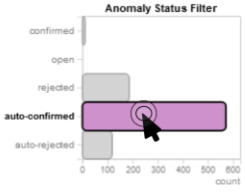
Selected anomalies are colored purple

While unselected anomalies are shown in gray.

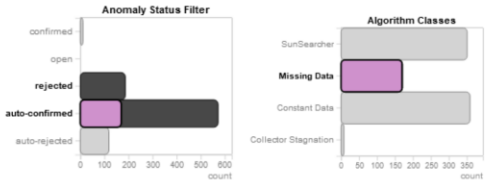
To filter out anomalies of a specific category, selections can be made by clicking on the name...

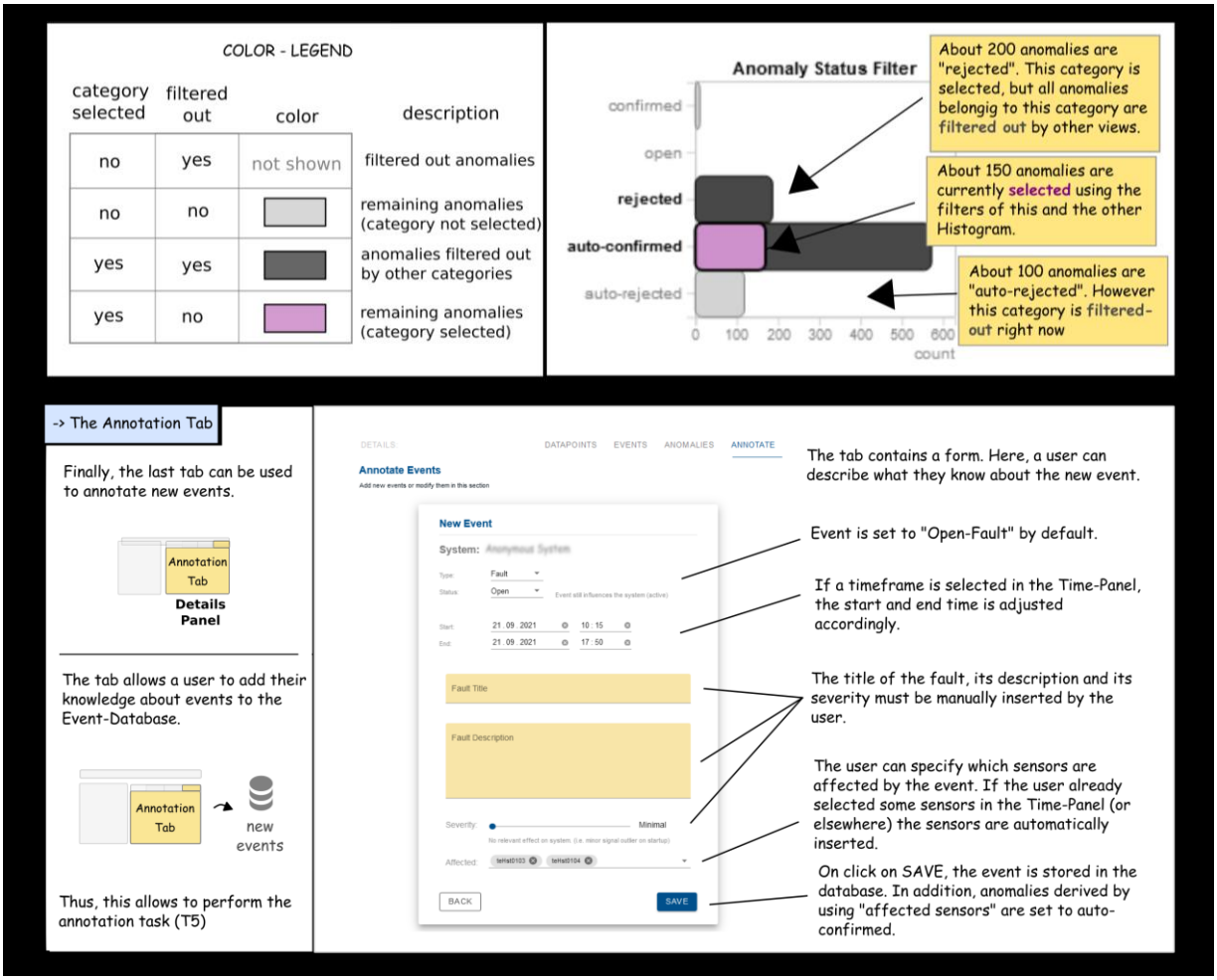


... or clicking on the bar's area.



And combining multiple filter in both histograms is possible too by toggelingn multiple bars.





6.3 Design Decisions

SunScreen was developed using rapid prototyping (see Methodology in section 2), and many feedback sessions with domain users were carried out to improve its design. This section discusses these results and explains why certain visual encodings were used for certain parts of *SunScreen*.

Why split into *Time-Panel* and *Detail-Panel*?

The main idea for splitting the screen into two sections was to reserve the main area for the most critical views, where the user spends most of their time, and a smaller area for details on demand. Hence, the Time-Panel is supposed to show the central aspect of the data, while the Details-Panel provides additional information if needed. A similar layout strategy has also been used by Liu et al. [4], displaying diagnosis details on the right side while reserving the main area for their central network-graph view. Similarly, Zhou et al. [11] display details on anomalies and selected sensor values on the outer right as well.

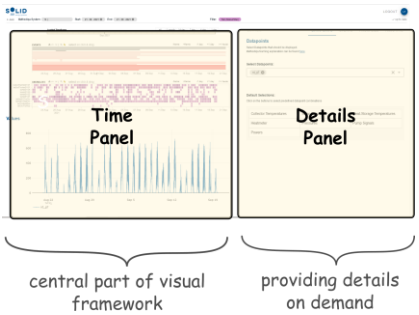


Figure 10: SunScreen is split into the Time Panel and Details Panel. The left part is supposed to show the central aspect of the data, while the right part provides additional details on demand.

Why focus on time?

The Time-Panel is the central element of *SunScreen* and shows the data (events, anomalies, sensor values) versus time. However, we also investigated other options in the early stage of the project (see Figure 11). For instance, one prototype focused on the sensors' correlations, displaying them in a network graph similar to Liu et al. [4] or Shi et al. [5]. Another one used the position of the sensors as the central element of the view, similar to Zhou et al. [11] and most SCADA systems. Yet another approach used a list of sensors, rated by their anomaly score, as an entry point for further exploration. Especially the position-based approach was very well received by the domain users. However, they still preferred the time-based approach used in the current version of *SunScreen*. They argued that understanding the temporal behavior of the system is the most critical aspect of the data. That is because some latent faults only show at certain times of the day and because the temporal behavior is also essential to evaluate detected anomalies.

One user also pointed out that the framework should support analyzing multiple plants (goal G5) and that the adaption time needed after changing from one plant to another should be as low as possible (G1). He argued that much more time is needed to comprehend new network graphs than to adapt to a new heatmap or line charts.

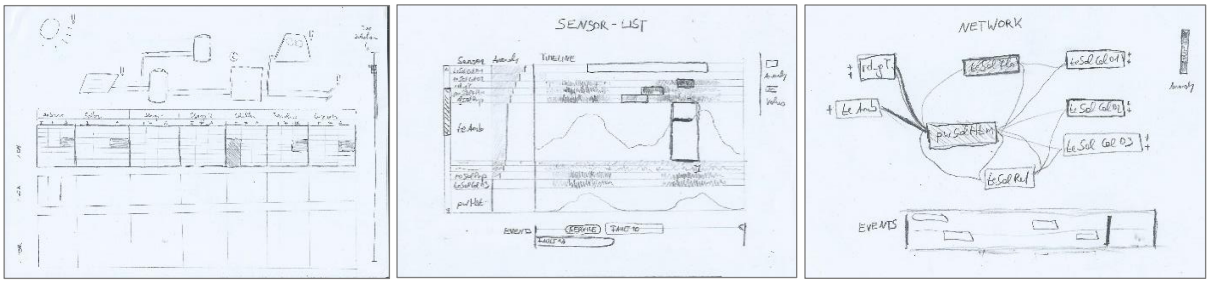


Figure 11: Three examples of early design prototypes using different parts of the data as their central element. The left prototype is inspired by Zhou et al. [11] and focuses on the sensor positions. The middle one is inspired by Live-RAC [39] and focuses on sensors and measurements. The prototype on the right is inspired by Shi et al. [5] and focuses on sensor relations.

Why not use the sensor positions in other views?

As discussed in the data abstraction, the sensor positions are vital to comprehending the system's behavior (see section 4). They play a critical role in interpreting the sensor data during evaluation (T2), manual fault detection (T3), and diagnosis (T4). In addition, they have been identified as a vital part of gaining an overview of the system (T1). Nevertheless, the sensor positions are currently not used by *SunScreen*. The reason is that the sensor positions do not change over time, and thus a simple printout or image of the system hydraulics is sufficient to support most of the abovementioned tasks. Although this is only an interim solution, using the sensor positions was postponed, and more time was spent on features with higher priority instead.

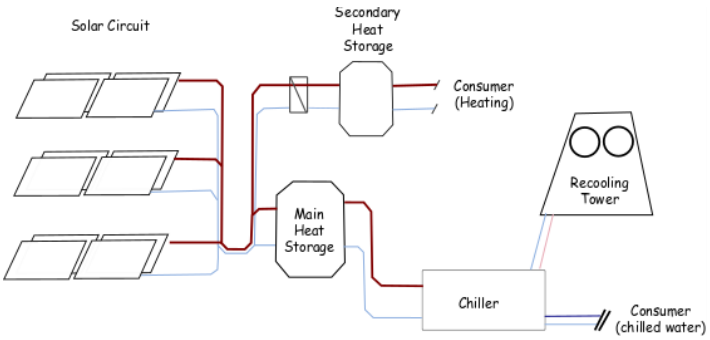


Figure 12: Sketch of a system layout. As the system's hydraulics rarely changes, a simple printout is sufficient to support most FDD tasks identified in the task abstraction in section 4.2. While there is some potential in combining the layout information with measurements and anomaly data, this feature has been postponed because of the limited development time.

Why use a *Timeframe-Selector*?

The Timeframe-Selector is located at the top of the Time Panel and tells the user what time period they are currently looking at. However, less space is available for the other views as a result. The feature was challenged by showing domain experts prototypes with- and without the *Timeframe-Selector.* The feedback showed that all domain experts valued this feature as it gives them more control in selecting time periods and provides an overview of what is currently visible in the views below (see Figure 13).

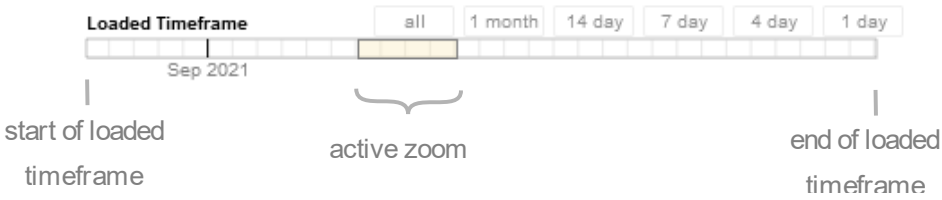


Figure 13: Screenshot of the Timeframe Selector. It shows the user what timeframe is loaded in the background and what zoom is active. In addition, they can alter the time period that is currently zoomed in by clicking on the buttons on the top right.

Why put the Time-Panel views on top of each other?

Each view in the Time Panel shows one entity of the data (events, anomalies, sensor values) versus time. As they all display the same time period, it makes sense to use a shared x-axis and plot them next to each other. This setup allows the users to compare the temporal behaviors of each data entity more easily. The order of the views is based on the workflow of the user. Hence, views needed earlier are displayed on the top of the screen, as users tend to scan a screen from top to bottom [40] (see Figure 14). Thus, events are displayed on top as they are needed for gaining an overview of the system (T1), followed by checking the anomalies (T1, T2), and finally, evaluating the system data (T4, T3).

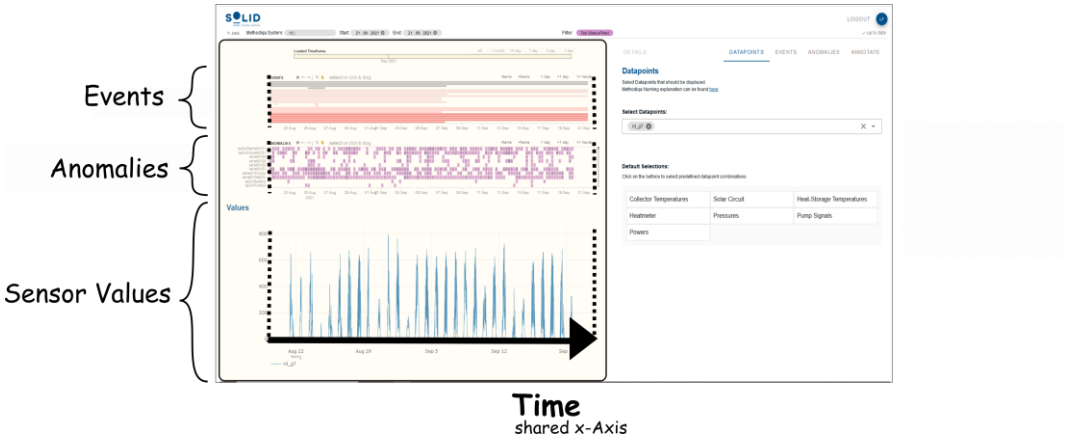


Figure 14: The Time-Panel shows events, anomalies, and sensor values versus time. The vertical layout and the shared x-axis allow the user to compare the temporal behavior more easily.

Why not show anomalies and sensor values in the same graph?

SunScreen displays anomalies and sensor data in subplots with a shared x-axis. Some authors instead chose to combine the data through encoding. For instance, Janetzko et al. [13] display the sensor data and highlight the anomalies using the color channel (either through different saturation or hue). One early prototype also used this approach, showing the sensor values of all sensors using resizable graphs with semantic zooming similar to LiveRAC [39]. This idea was ruled out as the domain users assumed that too much time would be spent resizing and zooming in on the graphs. The domain experts also argued that sensor values are essential to investigate details, but not all sensors are needed at once. Instead, they reckoned that separately showing sensor values and anomalies would provide a better overview (see Figure 15).

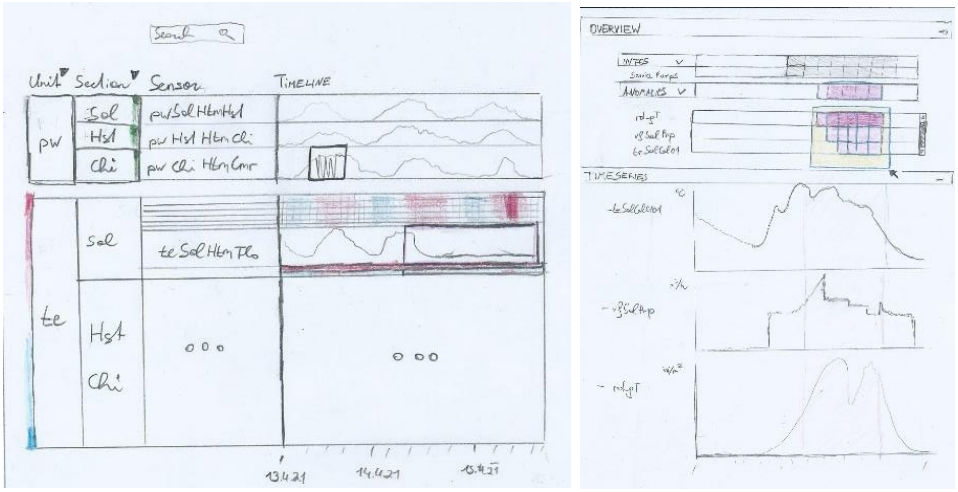


Figure 15: LEFT: Example of an early prototype inspired by LiveRAC [39] and Janetzko et al. [13] displaying the sensor values of each sensor (y-axis) versus time (x-axis) using resizable subplots. Each lane also shows the unit and position of the sensors, positioning similar sensors next to each other. The measurement data is shown using line charts or heatmaps, depending on the size of each lane. The black borders are used to highlight anomalous sensor data.
RIGHT: Another prototype - inspired by Visplause [8] and showing anomalies and sensor data in separate views - received better feedback from the domain users.

Why not show anomalies and events in the same graph?

Similarly, *SunScreen* also displays events and anomalies in different subplots. The choice to separate them is based on the different information they contain. The events are secured information from the domain user, while the detected anomalies must be verified first. The difference also shows in the relationship to sensors: Anomalies are tightly connected to sensors, as their measurement values are used for fault detection. In contrast, events might not be connected to sensors at all (e.g., in the case of a non-intrusive service), or affected sensors might only be identified during analysis sessions. Thus, the decision was to show events and anomalies in separate graphs.

Why use Gantts to display events?

The event data is displayed using Gantt charts - encoding the event's start, end, type, and severity. Other early ideas were based on line charts or heatmaps to show this information. These prototypes aggregate the data, showing the number of events or severity versus time. However, these ideas were ruled out because the domain user needs information about each individual event instead of aggregated information. In contrast, the Gantt chart provides the user with exactly the information he/she needs. Namely, the start (start of the Gantt), end (end of the Gantt), duration (length of the Gantt), type (hue of the Gantt), severity (saturation of the Gantt), and the title of the event (embedded as tooltip). By positioning the Gantts on lanes based on the type and severity of the event, the visual clutter is also minimized (see Figure 16).

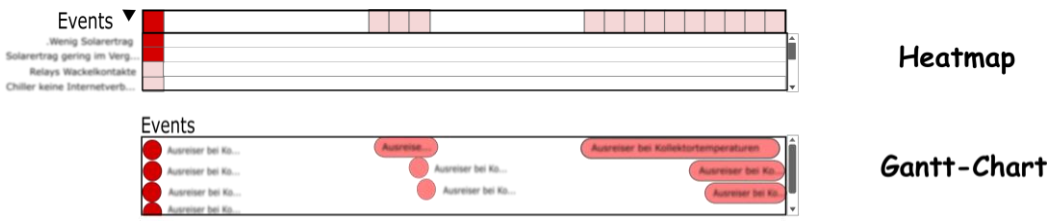


Figure 16: Sketch for comparing heatmap and Gantt charts to display system events. The Heatmap allows reading through all events, as the titles of the events are listed on the left side. However, it uses much more vertical space to display the data. In case too many events need to be displayed, a scrollbar must be used. In contrast, the Gantt chart on the bottom compresses the data and shows more events per screen area. In addition, titles are closer to the positions of the Gantts.

Why use a heatmap to display anomalies?

The Anomaly-Heatmap supports the user in identifying anomalous sensors and detecting correlations between anomalies. Especially the temporal evolution of the anomalies is essential to the user. Hence, the graph should encode the number of anomalies of each sensor versus time. Using heatmaps to display this information is thus both practical and intuitive, as they allow encoding time (x-axis), sensor names (y-axis), and the number of anomalies (color channel) intuitively. This choice is backed up by Visplause [8] and Visplore [9], which successfully used similar encodings and inspired our Anomaly-Heatmap. Nevertheless, other options that were considered are line charts, horizon graphs, and Gantt charts.

Line charts and horizon graphs were ruled out because they introduce unnecessary clutter. Both approaches use the strong position channel (y-axis) to encode the number of anomalies. However, the exact number of anomalies is unimportant to the user. Instead, they only want to know if a sensor is anomalous or not. In contrast, a heatmap is better suited to display this information by using a color scale and leaving cells empty if no anomalies are detected (see below). This encoding also allows for identifying patterns of anomalous sensors more easily.

On the other hand, Gantt graphs were ruled out because they require too much space. Many anomalies might get detected for the same sensor at the same time. Thus, overlapping anomalies would need to be displayed in separate lanes. This way, many lanes would be needed to display all anomalies while not much relevant information is introduced to the user. In comparison, a heatmap provides a much more intuitive way to interpret the data by aggregating the anomalies (see Figure 17).

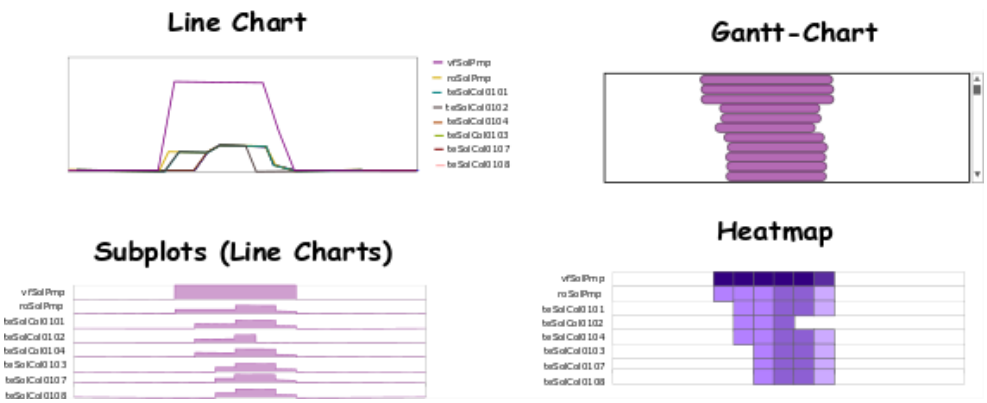


Figure 17: ideas for displaying anomalies. Line charts (left side) use the strong position channel to encode the number of anomalies. However, the exact number of anomalies is less critical to the user. In addition, it is hard to identify correlations between anomalous sensors. In contrast, using Gantt-Charts to display every anomaly (top right) needs too much y-space. Finally, we stuck with heatmaps (bottom right), as they best allow us to spot temporal correlations between anomalies of different sensors.

Why color-encode the number of anomalies?

The arguments above raise the question of why to encode the number of anomalies at all. This information is indeed less important than whether an anomaly was detected at all. However, many anomalies reinforce that a fault rather than a false-positive alarm is raised, as many algorithms are triggered independently. Hence, *SunScreen* intentionally encodes this information in a less efficient channel, using a linear color scale (see Figure 18). Hence, the user can distinguish how many algorithms found an anomaly. However, the scale is clamped at ten anomalies, as this already means that a multitude of algorithms detected a problem. Limiting the color scale also makes sense, as only 6 to 12 color levels can be distinguished visually [16]. In addition, cells are not drawn if no anomalies are detected. This encoding reduces visual clutter as these cells are uninteresting to the user. The same technique is also used by KnowYourEnemy [10].

Figure 18: Sketch showing how the color encoding for the anomaly heatmap works. Color is assigned based on how many anomalies get detected for the same sensor at the same time. No anomaly means that the sensor is unsuspicious and thus can be ignored. Hence, cells with no anomalies are hidden. In case anomalies are detected, a linear color scale is used.

Why use line charts to display the sensor values?

Line charts were used in almost all related work to display the details of the sensor data. Nevertheless, early prototypes also explored alternative encodings for the measurement value of the sensor data (S2). For example, we analyzed Spiral-Graphs, Heatmaps, and Pixel-based views using the color-channel and horizon graphs using a combination of position- and color-channel were reviewed. However, after a short discussion with a domain user, we opted for the traditional line charts, as they are well understood by the domain experts and allow them to identify the measurement values accurately (see Figure 19).

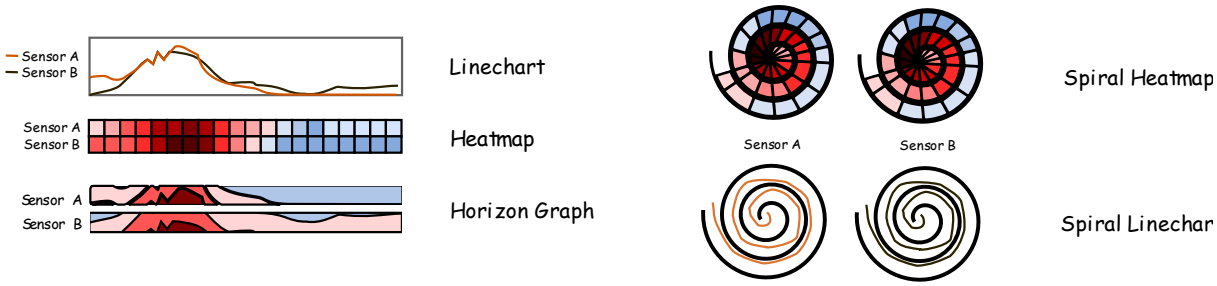


Figure 19: Sketches of different potential encodings for displaying the sensor data.

Why mix juxtaposition and superposition in the line charts?

In the Time Panel, measurement values of sensors with the same unit are shown in the same plot, while sensors with different units are shown in separate subplots. This was done to better support comparison. For example, the user might want to select all collector temperatures and check for outliers. By showing the data superimposed, this is very easy. The same is true for analyzing the heating up of a heat storage, for example. Here, the user might check the temperatures of the solar flow and different positions in the heat storage and see how the storage temperatures gradually rise. These comparison tasks are much more manageable if values are shown on the same plot. However, if too much different data is shown, the lines are harder to distinguish, resulting in a "spaghetti plot" [41]. Hence, the choice was made to separate sensor data with different units, as detailed comparisons are unlikely (see Figure 20).

Why Tabs for the *Details Panel*?

The requirement analysis (section 4.4) shows that not all parts of the data are needed at once. Instead, solving different tasks might require very different parts of the data. Showing all the data at once would need a lot of space and overwhelm the user with information. Thus, tabs were used to access details on demand while hiding unrequired information. This way, the user can act flexibly but can still focus on the data they need at the time only (see Figure 21).

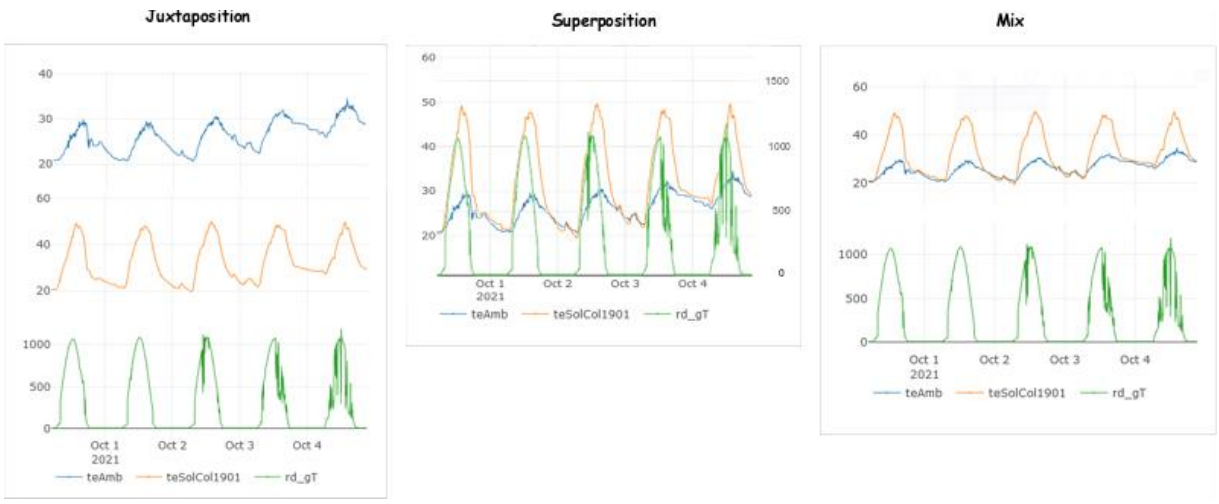


Figure 20: Sketch of three different line charts showing the same sensor data of three sensors. It shows how the irradiation (green) heats the fluid in the collectors (orange). At night, the collector temperature (orange) cools down to ambient temperature (blue).
***LEFT**: Lines are shown juxtaposed. The daily patterns are easy to spot, and each line can be followed effortlessly. However, a lot of space is needed to display the data, and it is hard to see that the collector temperature (orange) cools down to ambient temperature (blue) at night.*
***MIDDLE**: sensors values are shown superimposed. This encoding makes comparisons between sensor values easier and needs less space to show the data. For instance, it is easy to see that the collector temperature (orange) and ambient temperature (blue) are almost the same at night. However, if too many lines are drawn, it is hard to distinguish them, and it is harder to see individual trends.*
***RIGHT**: Lines are shown superimposed for sensors with the same measurement quantity and juxtaposed for sensors with unmatching units. This way, individual lines can be followed easily while allowing a good comparison between similar sensor measurements. For example, in this view, it is easy to identify the daily trend of the irradiation (green), its effect on ambient temperature(blue) and collector temperature (orange) during the day, and the cooldown of the collector temperature(orange) to ambient temperature (blue) at night.*

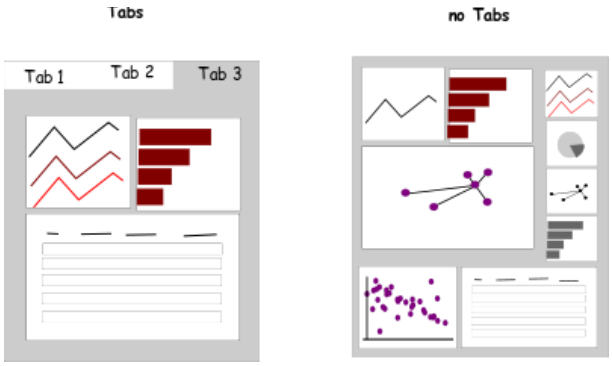


Figure 21: Sketch to illustrate why SunScreen uses tabs. Assuming that only some views are needed to solve the users' current task, using tabs allows us to show them in greater detail (left – all relevant graphs for the current task are shown in Tab 3). If all views are shown at once instead, this may create unnecessary visual clutter and distract the user (right – only the red-colored graphs are relevant to the user, but they might get distracted by other graphs).

Why use these specific Tab categories (Datapoints, Events, Anomalies, Annotation)?

The driving idea was to use the most intuitive and self-explaining categories to group the views into different tabs. Thus, the goal was to make it easy for the user to click on the correct tab [42].

In the current version of *SunScreen*, the categories are based on the data entities: events, anomalies, and sensors. These categories are pretty straightforward to the users, such that it is easy to click on the correct tab if more information about some data entity is needed. That is also consistent with the Time Panel, where the same data entities are shown versus time.

Some might note that we chose "Datapoints" for the tab associated with the sensors. As an alternative, the name "Sensors" was discarded as it might mislead users to think that more information about the sensors (i.e., the type of the sensor, its location, or description) will be provided. Similarly, the name

“Sensor-Values” might have implied that statistics about the sensor data are shown in the tab. Hence, we chose “Datapoints” as this more intuitively conveys that the tab can be used to select and deselect data points shown in the Sensor-Values panel.
In addition to the tabs associated with the data entities, one tab is reserved for the annotation task. This was done as it needs a considerable amount of screen space and is only used during annotation (T5).
An alternative to the data-entity-based approach would have been to use a tab for each identified task (T1-T5) instead (see Figure 22). However, tasks like evaluating anomalies (T2) and diagnosing faults (T4) are very explorative and might need various graphs to solve all questions. Putting all of them in one tab would result in an overcrowded design (see Figure 22).

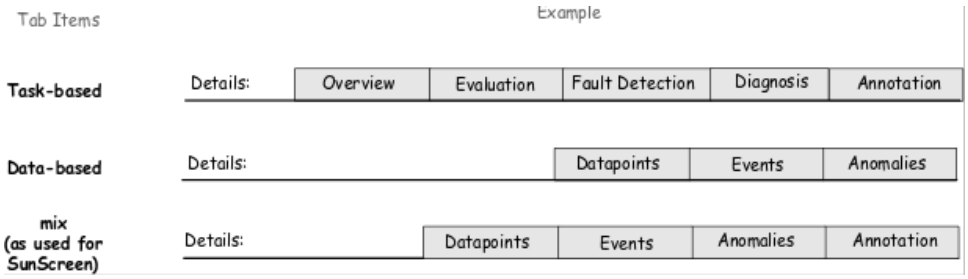


Figure 22: Sketch of grouping options for the Tabs in the Details Panel. The task-based approach allows guiding the user through each task. However, some tasks like Diagnosis require analyzing the data from many perspectives. Putting all required graphs into one tab would thus lead to an overcrowded design. Instead, the data-based approach provides details about each of the data categories. During diagnosis, the user can thus switch between tabs to get more details about each category, while an overview is provided in the Time Panel. This way, the user is flexible in its analysis and can quickly identify which tab to click to get to the data they need. Finally, the last approach adds a tab for annotating new events, so enough space is available to support this task.

7 Evaluation

This section contains the evaluation results of *SunScreen*. The first part deals with the outcome of the usability tests, while the second part presents the results of the Use-Case.

7.1 Usability Tests

As described in section 2, usability tests were conducted with five domain experts from SOLID, including one target user. After a short (about 20 minutes long) introduction to *SunScreen*, participants were asked to perform some predefined tasks or explore *SunScreen* on their own. To gain more insights into the participants' thoughts, they were told to describe their interactions and what they intended to do by speaking aloud. After working with *SunScreen* for one hour, they were asked to fill out an unmodified System-Usability-Scale (SUS) questionnaire [17, 18, 19]. It includes ten statements about the system's usability that can be rated using a Likert Scale with five categories (strongly agree to strongly disagree). Using a scoring mechanism, the SUS provides scores between 0 and 100 which may help interpret the usability of the visual framework or compare it to other ones.
After filling out the questionnaire, the participants were asked to give overall feedback about *SunScreen* in a short semi-structured interview (about 5 minutes).

The results from the SUS questionnaires can be seen in Figure 23. Individual scores range between 50 and 72.5, and the average score is 62.5. Based on empirical studies done by Bangor et al. [19], this indicates only moderate usability of *SunScreen*. Statements that received low scores were “I think I would need the support of a technical person to be able to use this system” and “I would imagine that most people would learn to use this system very quickly”. All other statements were rated as good or at least with average scores by all participants.

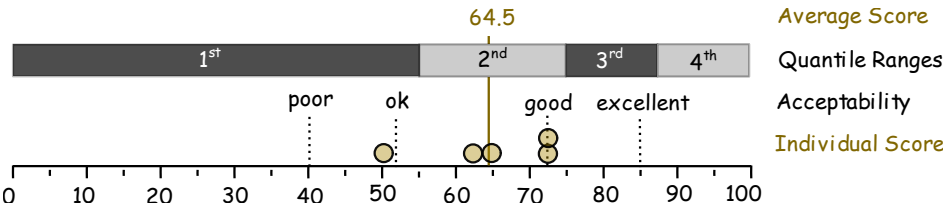


Figure 23: System Usability Scale results for SunScreen. The scores of each participant are displayed as yellow circles, while the average value is displayed as a yellow line. In addition, quantile ranges and acceptability for SUS tests based on empirical studies [19] are displayed to better interpret the results.

Even though the individual scores for each participant differed, the feedback from the interview was relatively homogeneous. Apart from some bugs and inconsistencies that the participants discovered, the following issues are the most relevant:

> **Missing knowledge / "overwhelming information".** Most users reported being overwhelmed with information at some point in their interaction with *SunScreen*: "*Too much information at once … events, anomalies … one cannot solve this in one session*". One domain user remarked that "*the problem is not the tool, but that there are very few people that can interpret both the sensor data and the algorithm results.*". Using these types of anomaly detection algorithms is still new to most domain users who participated in the usability study. Hence, being tasked to understand the algorithm results and interpreting the system's behavior at the same time was a difficult task. It indicates that the tool should provide better guidance and overviews for the user and better ways to make the algorithms understandable. Another user mentioned that "*one has to grow into it*", indicating that more extensive training might also result in better usability of *SunScreen*.
>
> **"Too many anomalies".** Another issue was the sheer number of anomalies found by the algorithms. Because they were not optimized, the algorithms were very sensitive and resulted in many wrongly identified anomalies (false positives). This was done deliberately to check if *SunScreen* is good enough to compensate for bad fault detection algorithms. However, users were frustrated when faced with thousands of anomalies and an extraordinary rate of incorrect suggestions (around 90% false positives). They reported that "*there are too many anomalies. They need to be pruned beforehand*" and that "*there are thousands of useless anomalies, and only maybe a hundred of them are correct. I don't have enough time to check them all*". Hence, *SunScreen* cannot compensate for such a high false-positive rate of anomalies. Instead, the anomaly algorithms either need better accuracy or *SunScreen* must be improved to identify and filter out incorrect anomalies.

In addition, domain experts identified some less severe issues during the evaluation. Figure 24 shows the number of issues reported by the user, categorized based on the impact on *SunScreen's* usability. This categorization was done by the author based on the participants' feedback and is subjective. The two most important issues are the ones reported in the text above. Some of the issues could be fixed shortly after the usability test. For instance, the number of false-positive anomalies was reduced by discarding the *Collector-Similarity* algorithm class, which yielded the most erroneously raised anomalies. The remaining issues could not be solved immediately and are left for future work.

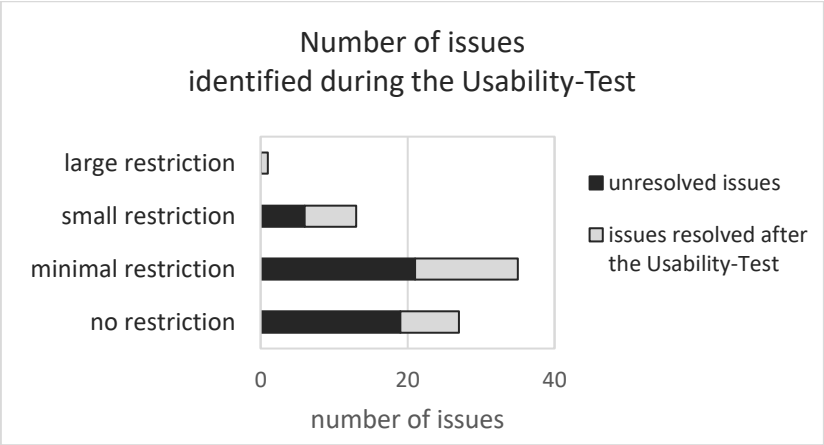


Figure 24: Number of issues that the participants identified during the usability test. The issues are grouped based on their influence on using SunScreen. Hence, "large restriction" means that the identified issues of this group largely restrict the usage of SunScreen. In contrast, "no restriction" means that these issues have no influence on using SunScreen as intended and are merely ideas for improvement. Issues colored in light grey were resolved shortly after the usability test, while unresolved issues were marked in darker colors.

7.2 Use-Case Scenario

After fixing some of the issues identified in the usability test (see above), a use-case scenario has been carried out with one domain user from SOLID. Unfortunately, the target user was not available for the use-case scenario. Nevertheless, the domain user who carried out the scenario has similar knowledge about solar thermal systems and is skilled at analyzing system data.

In two sessions, the participant was tasked to check the status of a real-life system for two arbitrary months in the dataset. Not being a target user, he had no previous information about what happened at the plant during that time. In each session, the first part of the meeting was spent explaining *SunScreen* in further detail, giving tips on how to use it best to detect and analyze anomalies. This was mainly done to compensate for missing experience with anomaly detection algorithms. In addition, as the participant is not a target user, it was also critical to discuss some parts of the monitoring workflow in more significant detail beforehand. After the introduction (about one hour long), the participant was tasked to find as many faults as possible in about half an hour.

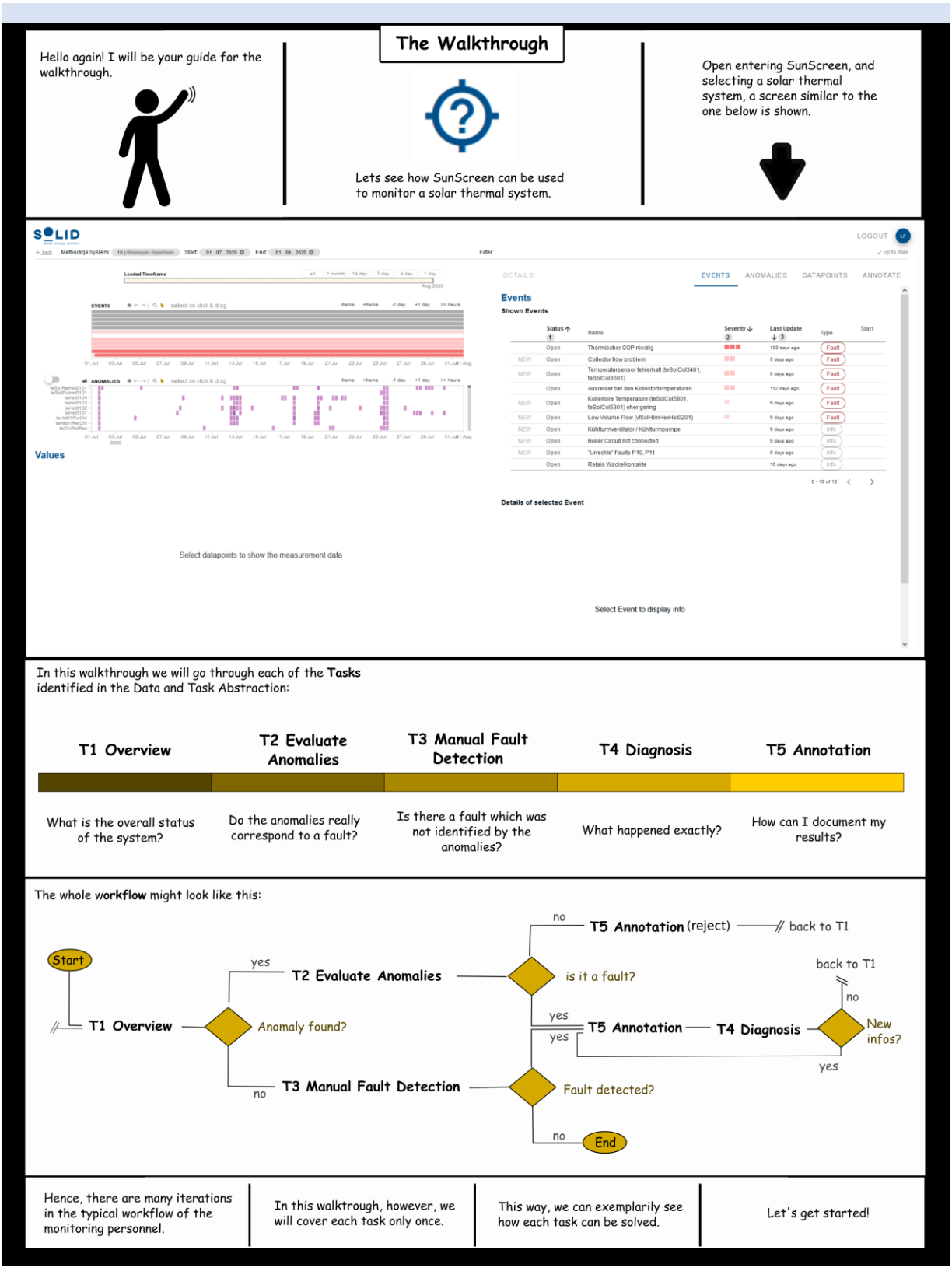
In the first session, the domain user mostly tried to get a better feeling for *SunScreen*, spending most of the time understanding how the visual framework works instead of identifying faults. In addition, some bugs and usability issues delayed his work. The feedback thus was very similar to the usability test, stating that the interaction with *SunScreen* is sometimes overwhelming. However, the reduction of anomalies due to better algorithms received positive feedback.

In preparation for the second session, some remaining major bugs were resolved. This time, the domain user spent his time following a similar workflow as described in the walkthrough in section 7.3, reading through the events and analyzing some anomalies. As a result, two faults were identified – one of which was not yet discovered by the monitoring personnel. However, both faults were minor, with little to no influence on system performance. The domain user thus argued that *SunScreen* might shift too much focus on unimportant faults. On the positive side, he noticed that he felt way more "secure in using *SunScreen* compared to the last time, and now [he] understand[s] what happens in the background". In addition, he mentioned that *SunScreen* is "nice for looking at the data" and that it is critical to have access to the annotations when analyzing and diagnosing the system. He reported that "link[ing] and store[ing] and share[ing] information is the basis for improving [the quality and speed of monitoring]".

A few weeks later, the domain user reported that he had used *SunScreen* again, exploring the system data to understand a particular system behavior as part of a research project.

7.3 Example Walkthrough

This section contains an example walkthrough of *SunScreen*. As before, a data comic is used to demonstrate the interactions with the visual prototype. The walkthrough was done after the Usability-Test and the Case-Study, using the latest version of *SunScreen*. Hence, some issues identified in the Usability-Test were already fixed, and the number of anomalies was drastically reduced in comparison. The walkthrough was performed by the author. However, it uses real-life data and thus is representative of how *SunScreen* can support the target user during their work.



T1 Overview

The first task is to get an overview of the system.

Q1: What is the overall status of the system?

And we want to get that overview based on:

Events	Q2: What events happened recently?
Anomalies	Q3: Are there any anomalies?
System-Hydraulics	Q4: What is the layout of the system?

Answering all of these questions will also answer our main question Q1

Q1: What is the overall status of the system?

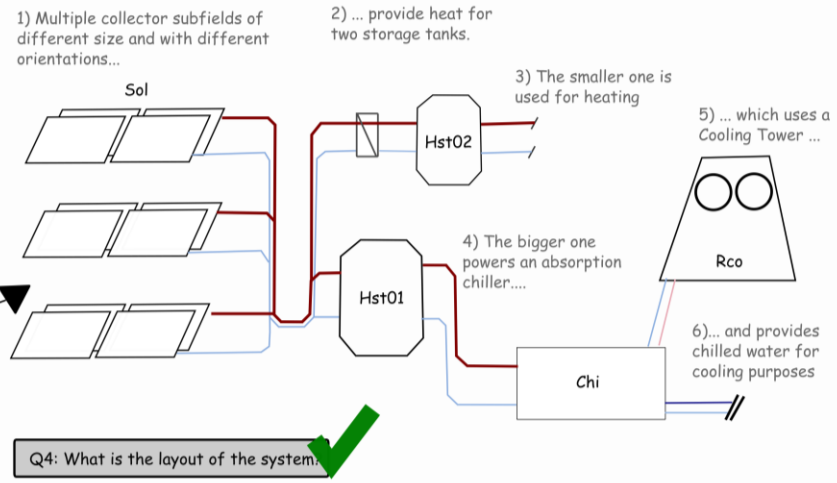
Q4: What is the layout of the system?

We want to know the systems hydraulics to better understand event descriptions and the behaviour of the system. This will help us in all subsequent tasks.

Unfortunately, displaying the layout is not yet possible with SunScreen. ✗

Nevertheless, it is easy to compensate this, using a printout of the system hydraulics.

Here is a sketched overview of the solar thermal system:



Q2: What events happened recently?

Knowing what events recently happened has many benefits for the monitoring personnel.

(1) It ensures that no time is spent on identifying faults which were already indentified by another user.

(2) It allows to make connections between events and anomalous data.

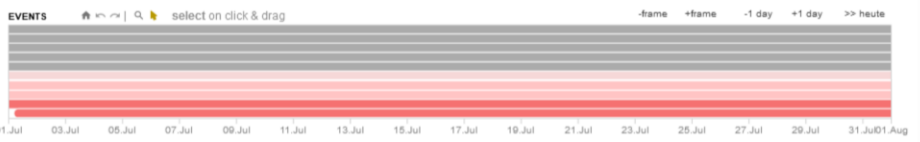
(3) It allows to gain a deeper understanding of the system's status and thus helps prioritizing work.

To answer the question, we take a closer look at the EventGantt in the Time-Panel.

This is how the EventGantt looks like for the time period selected.

Each bar represents an event.

Thus, about 10 events are currently known of during the selected timeframe.



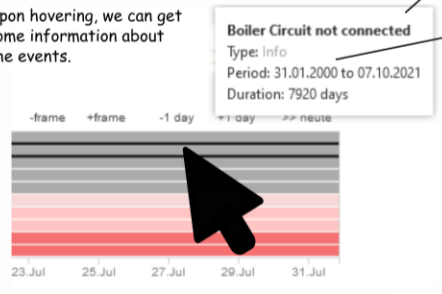
Most of the events span the whole width of the chart. This means that the events affect the system the whole time.

Some of these events may arise from open faults, which are not yet dealt with and were detected only recently...

Others are typically "unterminated" hints, which constantly inform the user about some interesting aspect of the data.

To get an overview of the events and benefit from the annotated information, we want to read through the events.

Upon hovering, we can get some information about the events.

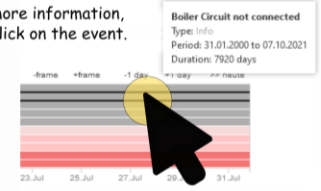


For example, the second event tells us, that the "boiler circuit is not connected".

It also shows that the event is active since very early on.

Hence, this is an example of one of the "unterminated" events as described above

To get more information, we can click on the event.



Clicking on a event will automatically open the Event Tab.

In the EventTab, all the information to the event is displayed.

The **description** explains that some sensors report misleading data.

And the **duration** tells us that this info is informative for the whole lifetime of the system.

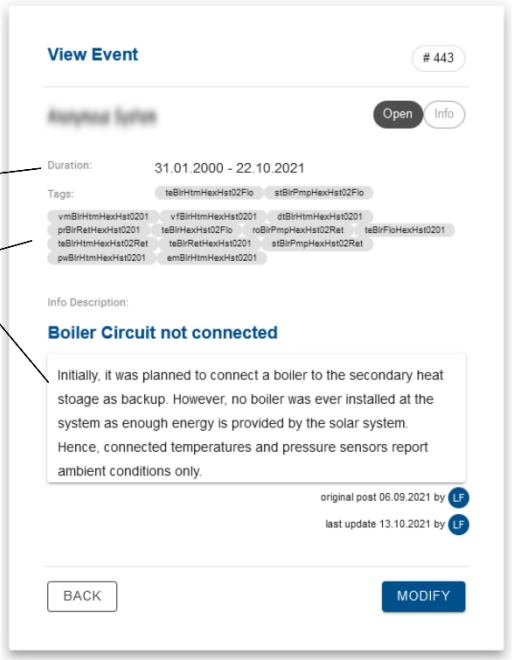
The **affected sensors** are linked, and their measurment values can get displayed by clicking on the Tags.

Importantly, the sensors are excluded from the Anomaly Detection, if they are tagged.

This is nice as the sensors report "bad" data only. Thus, otherwise many uninteresting anomalies would have been raised.

This way, the user (we) don't have to rediscover this anomaly again and again!

Thus, all the "Boiler - Sensors" are now excluded from anomaly detection as they are known to be non-informative.



In this way, all currently active events are checked.

One other event notifies that two redundant collector sensors appear to be broken.

The rest of the events are also relevant for the monitoring personnel, but less important for our walkthrough and hence omitted here.

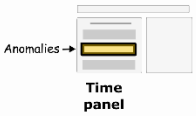
Thus, we conclude

Q2: What events happened recently?

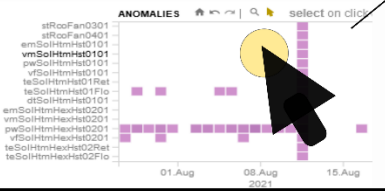
Some sensors are not informative!
Two collector sensores are broken!
(+ some other events which are not relevant for the walkthrough)

Q3: What sensors indicate anomalis? Are there any patterns?

with events and system layout checked, let's look for anomalies in the data.



To do so, we look at the Anomaly Heatmap.

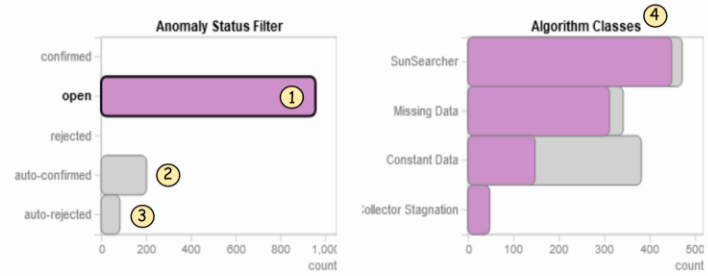


We also want to see the AnomaliesTab right away, beacuse it gives more information about the faults. The fastest way to access it is to click on the Anomaly Heatmap.

Now, the screen looks like this:

Currently, a lot of information is displayed on the heatmap, showing all the anomalies that were found in the timeframe.

To get a better grip on the data, we will check the AnomalyTab - more specifically the histograms on the top right.



(1) The left part shows us that only open anomalies are selected. This is exactly what we need right now, as confirmed or rejected anomalies are already dealt with.

So there are about 900 "open" anomalies we still have to check.

This selection results in only "open" anomalies to be shown in color on the heatmap, while auto-rejected and auto-confirmed anomalies are grayed out.

(2) The auto-confirmed anomalies correspond to events that have already been detected (e.g., the sensors tagged in the "Boiler not connected" event)

(3) The auto-rejected anomalies belong to algorithms that were found to yield bad results only and thus were disabled.

(4) The other side shows how many anomalies are detected by which type of algorithm.

SunSearcher is the Machine Learning algorighm.

Missing Data checks for missing data

Constant Data checks for constant data

Collector Stagnation checks for collector temperatures > 130°C

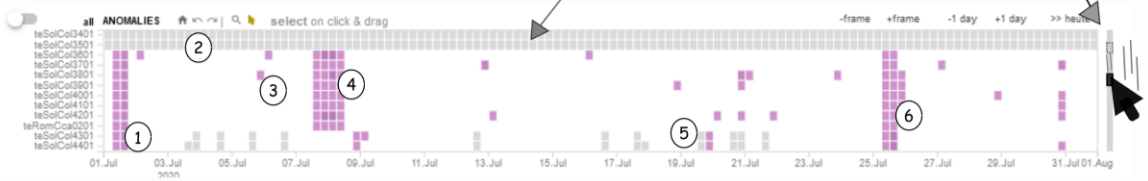
It is interesting that the collector stagnation checker detected faults, as this indicates some more severe problems.

(at this point, T2 Anomaly Evaluation might be started already)

In addition to the summary diagrams we can also take a look at the AnomalyHeatmap, to check the time-dependency of the faults. Depending on the temporal behavior of the anomalies, some rule-of-thumbs can be applied:

The cells are grayed out because these anomalies are "auto-confrimed". They correspond to the "broken temperature sensor" event we discovered earlier.

We use the scrollbar to scroll through all the sensors and their anomalies..



(1) **Vertically** spanning anomalies mean that a lot of sensors are affected (e.g., missing data). The likelyhood that the anomalies correspond to an event is high, as many algorithms detect a problem independently.

True-Positive-Rate: high
Severity: low to high (depending on width)

(2) **Horizontally** spanning anomalies indicate that one sensor (or group of sensors) has issues. They also have a high likelyhood that this indeed indicates a fault.

True-Positive-Rate: medium to high
Severity: medium

(3) Anomalies that appear **alone** tend to be data-outliers or correspond to less severe events. However, there can always be exceptions.

True-Positive-Rate: low
Severity: low to medium

(4) If only **some sensors** are affected at the **same time** this still indicates high likelyhood that an event occured at that time, influencing only some sensors.

True-Positive-Rate: medium
Severity: low to high

(5) If anomalies are detected for **one sensor multiple times** in a row, this may either indicate that the algorithm is too sensitive, or that the sensor indicates some problems.

True-Positive-Rate: low to medium
Severity: medium

(6) Combination of (1) and (4) indicates that an event occured and caused another event as direct consequence.

True-Positive-Rate: high
Severity: medium to high

After scrolling through all the sensors we now have a rough overview of the anomalies that occured at the system.

We now might check the anomalies, which indicate the highest potential for detecting a fault first.

Q3: What sensors indicate anomalis? Are there any patterns?

Multiple patterns detected (See above).
The anomalies with highest potential to detect a fault are the ones belonging to the Collector Stagnation Class.

As a result we solved all three questions needed to get an overview of the system.

Events — Q2: What events happened recently?

Anomalies — Q3: Are there any anomalies?

System-Hydraulics — Q4: What is the layout of the system?

And thus we also answered the more high-level question Q1:

Q1: What is the overall status of the system?

Events indicate some open issues.
Anomalies indicate some problems - espcially the Collector Stagnation Algorithm.

T2 Evaluate Anomalies

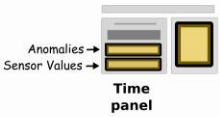
After we found some interesting anomalies in T1, now the task is to evaluate them to see if they correspond to a fault or not.

Using a greedy approach we will start investigating the most intriguing anomaly first.

That is, we want to know more about the Collector Stagnation Anomalies we spotted in T1.

To fulfill this task,
we want to answer the following questions:

To tackle these questions, we will rely on the AnomalyHeatmap, accessing more info via the AnomalyTab and check the sensor values in the SensorValues View

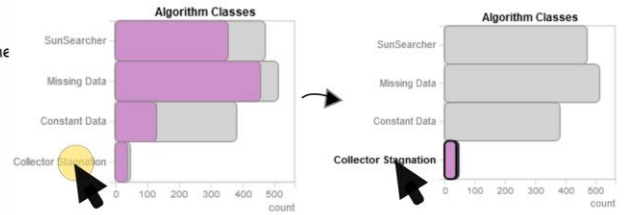


- Q5: Which sensors are correlated to the anomaly?
- Q6: What are the measurement values of the affected sensors during the anomaly?
- Q7: How does that differ from typical behaviour?
- Q8: Is this indeed a fault?
- Q9: Do some anomalies correspond to the same fault?
- Q10: Which algorithms detected the anomalies?
- Q11: Is the anomaly detection algorithm working in general?

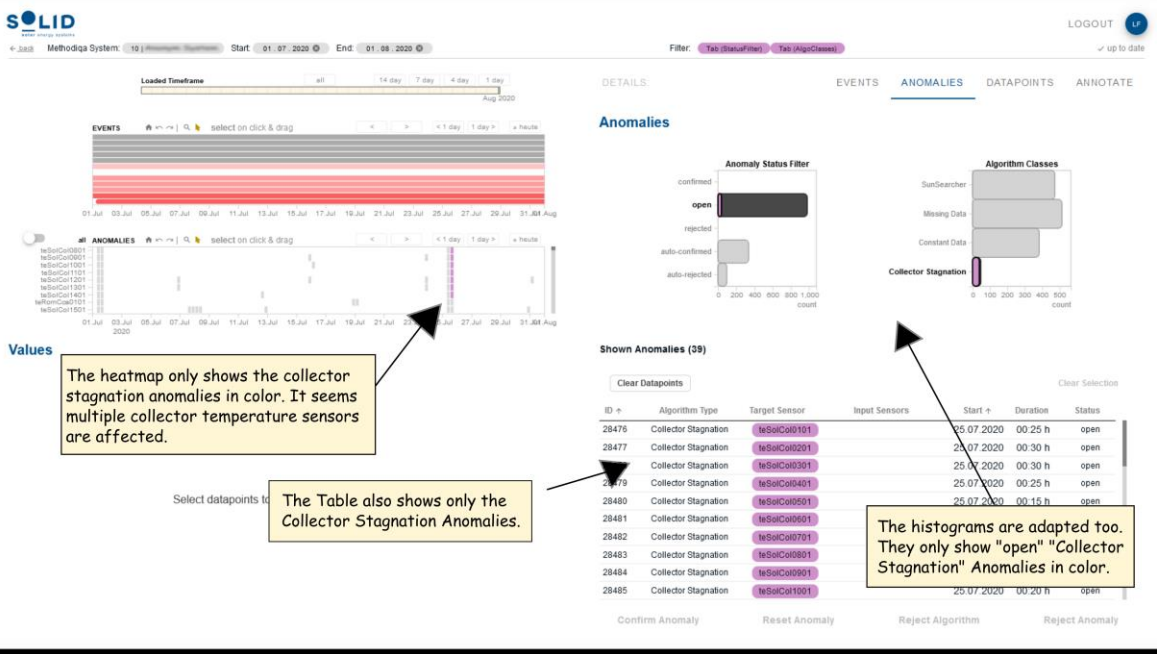
As described, we want to evaluate the Collector Stagnation Anomalies first.

To not get disctracted by other anomalies, we filter them out.

To do so, we apply the filter in the AnomalyTab by clicking on the category in the Histogram:



The other views adapt accordingly



The heatmap only shows the collector stagnation anomalies in color. It seems multiple collector temperature sensors are affected.

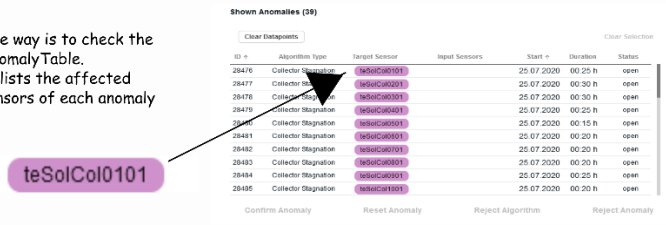
The Table also shows only the Collector Stagnation Anomalies.

The histograms are adapted too. They only show "open" "Collector Stagnation" Anomalies in color.

Q5: What sensors are correlated to the anomaly?

In a first step, we want to check which sensors seem to be affected by the anomaly.

One way is to check the AnomalyTable.
It lists the affected sensors of each anomaly

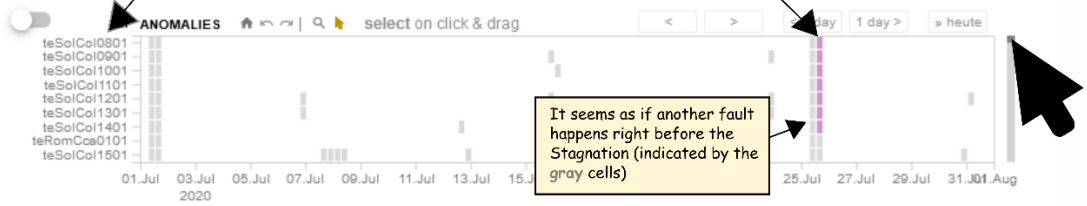


teSolCol0101

Another option is to check the AnomalyHeatmap.

On the left we can see the sensor names.

By scrolling up and down, and checking for the purple colored cells, we can identify all correlated sensors.



It seems as if another fault happens right before the Stagnation (indicated by the gray cells)

In both ways, one can see that multiple (but not all) collector temperatures sensors seem to be affected.

teSolCol0101 to teSolCol1401
teSolCol4301 to teSolCol6901

Thus, we solved our current question:

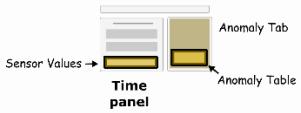
Q5: What sensors are correlated to the anomaly? ✓

Q6: What are the measurement values of the affected sensors during the anomaly?

To assess if the anomaly indeed detected abnormal behavior, we need to check the sensor values of the affected sensors.

To select the correct sensors, we use the AnomalyTable in the AnomalyTab.

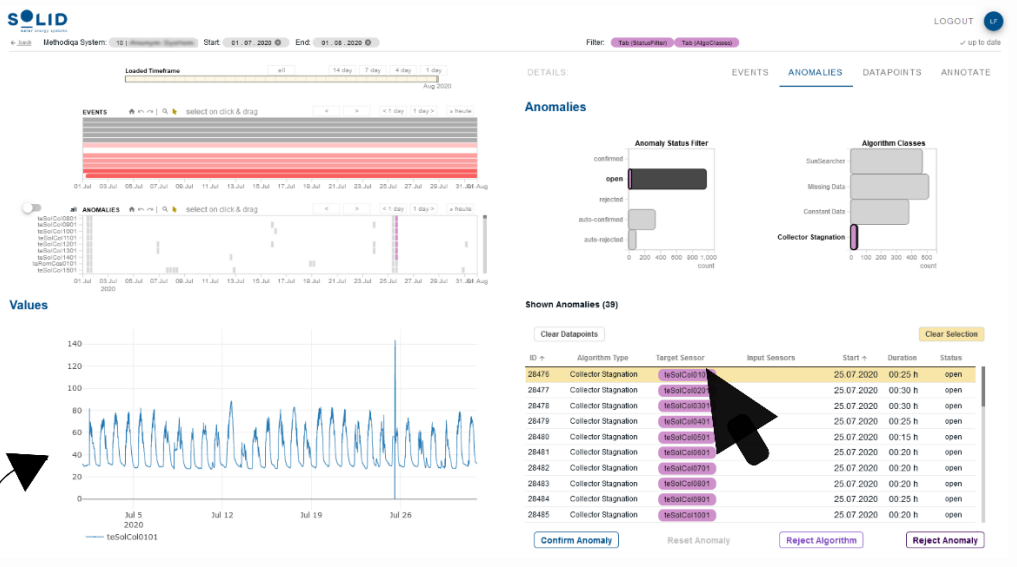
As an end-result we want to show their measurement values in the SensorValues view.



The AnomalyTable lists the affected sensors as tags.

Their measurement values can be displayed by clicking on the datapoint tags.

ID	Algorithm Type	Target Sensor	Input Sensors	Start	Duration	Status
28476	Collector Stagnation	teSolCol0101		25.07.2020	00:25 h	open
28477	Collector Stagnation	teSolCol02		25.07.2020	00:30 h	open
28478	Collector Stagnation	teSolCol030		25.07.2020	00:30 h	open



The SensorValues view now shows the measurement values of the selected sensor:

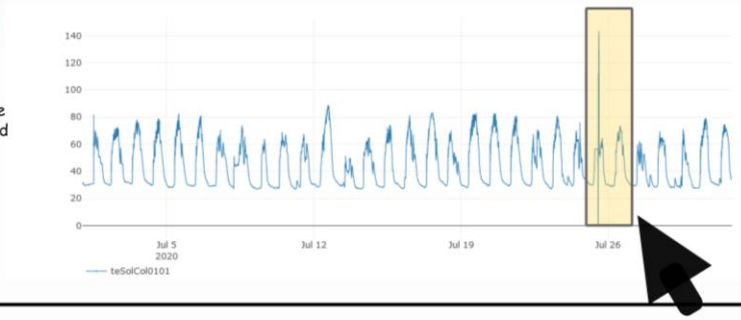
Thus we have completed our subtask, and can now evalute and interpret the measurement data.

Q6: What are the measurement values of the affected sensors during the anomaly? ✓

Q7: How does that differ from typical behaviour?

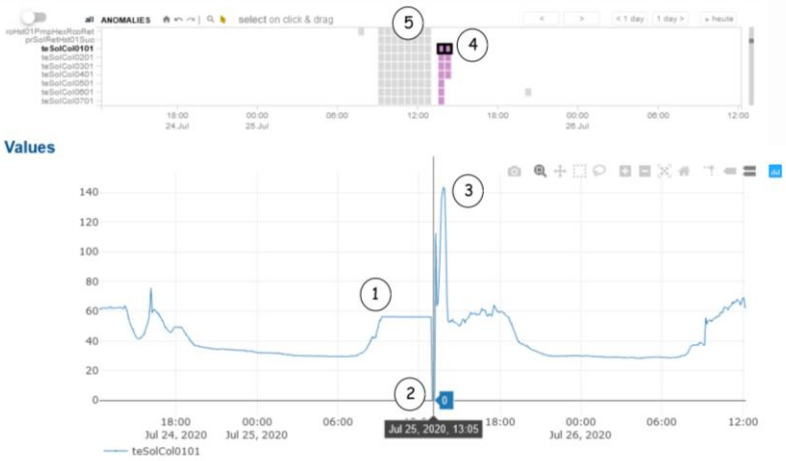
We can already see that there has been some abnormal data. On Jul 25th, there is a spike in the measurments, while the measurements before and after that day look similar to each other.

But not many details are visible yet. Let's take a closer look by zooming in with the zoom tool.



Multiple abnormalities can be seen at this graph:

1. The collector temperature is abnormaly constant between 8:00 and 13:00 o'clock
2. At around 13:00 o'clock the sensors reports 0°C.
3. Shortly after, the collector sensor reports rising temperatures up to 140°C
4. This spike (3) is also what caused the Collector-Stagnation Algorithm to raise an anomaly.
5. The constant data is reported by other anomaly algorithms (which are currently filtered out).



Thus, we found out that this sensor measurement really is abnormal compared to the typical periodical rising and cooling of the collector temperatures

Q7: How does that differ from typical behaviour? ✓

normal: periodic rising and cooling of collector temperatures
abnormal: constant data, 0°C, sudden spike and high temperatures

Q8: Is this indeed a fault?

We can also conclude from the findings above that a problem must have occured.

The measurement data with the large leap from 0°C to 140°C is very suspicous. It cannot be explained by the typical behavior of the system.

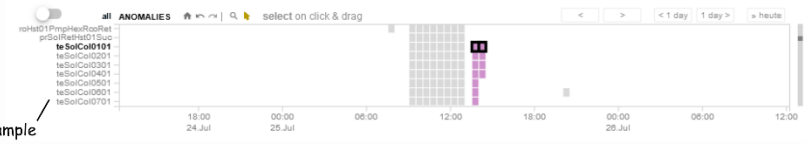
Q8: Is this indeed a fault? ✓

yes

Q9: Do some anomalies correspond to the same fault?

We also already know that multiple collector temperatue sensors are affected as well (see Q5).

These ones for example

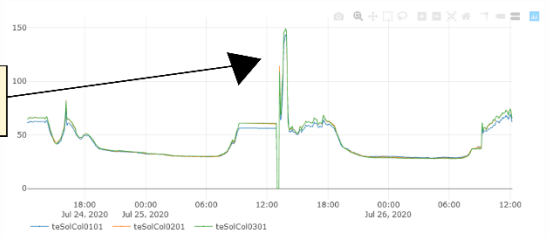


By clicking on their tags in the AnomalyTable...

ID	Algorithm Type	Target Sensor	Input Sensors
28476	Collector Stagnation	teSolCol0101	
28477	Collector Stagnation	teSolCol0201	
28478	Collector Stagnation	teSolCol0301	
28479	Collector Stagnation	teSolCol0401	
28480	Collector Stagnation	teSolCol0501	
28481	Collector Stagnation	teSolCol0601	
28482	Collector Stagnation	teSolCol0701	
28483	Collector Stagnation	teSolCol0801	

... the measurement data is shown in the SensorValues view

All lines show the same behavior.

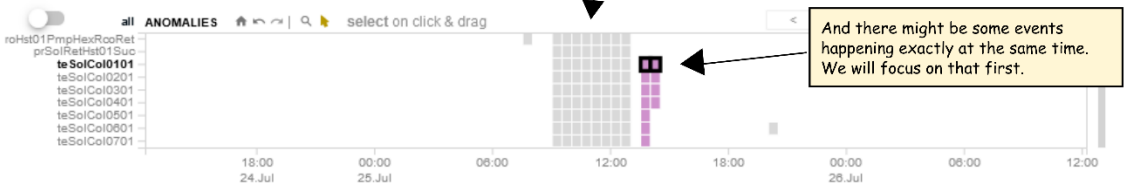


Thus, it is very likely that all of them were caused by the same event.

However, so far we only checked the Collector Stagnation Anomalies.
Maybe there are other algorithms giving some hints.

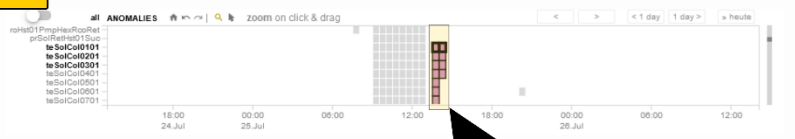
There is a large block of anomalies happening before the Collector Stagnation. We will check this later.

And there might be some events happening exactly at the same time. We will focus on that first.



Q10: Which algorithms detected the anomalies?

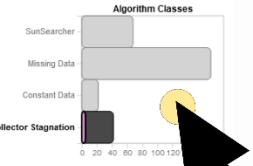
As subtask of Q9, we also check if another algorithm detected the same anomalous behavior.



To do so, we select the anomalous region in the Anomaly Heatmap using the Selection Tool.

... and reset the Collector Stagnation Filter in the Anomaly Histogram.

(Per click on any white space)



... now we can see that the SunSearcher Algorithm also found anomalies in the same period for the same sensors.

Hence, the Collector Stagnation Checker and the SunSearcher Algorithms detected some anomalies.

That multiple algorithms detected anomalies might further back up our assumption, that indeed an event occured.

Q10: Which algorithms detected the anomalies?

The Collector Stagnation Check and the SunSearcher

With the answer of Q10 we can finally also conclude Q9.

Anomalies were detected by the Collector Stagnation Algorithms and by the SunSearcher Algorithms and are likely to correspond to the same event.

In addition, the anomalies detected shortly before the Collector Stagnation might also correlate with the same event.

Q9: Do some anomalies correspond to the same fault?

yes, multiple anomalies seem to correlate to the same event

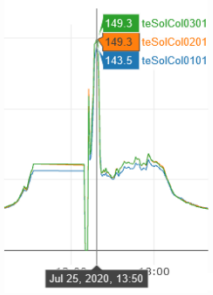
Q11: Is the anomaly detection algorithm working in general?

Finally, we want to assess if the algorithm (Collector Stagnation) is working in general.

The algorithm detected only one anomaly (as can be seen in the AnomalyHeatmap) ...

only one alarm in one month triggered by the Collector Stagnation Checker targeting teSolCol0101

... which indeed corresponds to a collector stagnation ...



... so all detected anomalies were indeed correct.

When considering this month only, this means that the algorithm has a true-positvie rate of 100%.

Thus we can conclude that the algorithm seems to work fine.

Q11: Is the anomaly detection algorithm working in general?

yes, it works fine.

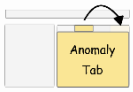
T5 Annotate

We just found out that the anomalies indeed correspond to some event (in T2 above).

To make this information available to others, we need to annotate it.

Thus, we want to switch to the AnnotationTab...

...but currently, the AnomalyTab is still active.



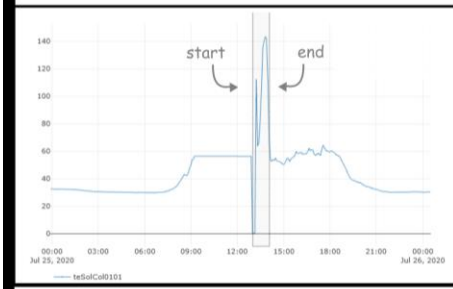
Next, the AnnotationTab opens, allowing us to insert all we know about the event right now.

1 First, lets insert the new event's status and type.

For the event-type we stick with the default, as the event we want to annotate really seems like a "Fault".

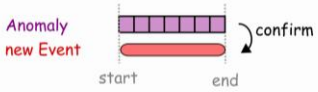
For its status we choose "Review" - ,meaning that the event is finished (no influence on the system anymore) but not understood yet.

2 Next, we can annotate the start and end time of the event.



To get a better feeling what timeframe is annotated, the selected period is shown in the SensorValues view as gray background.

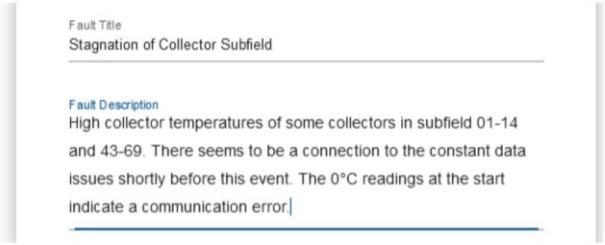
If the "Confirm Anomaly" button was used to switch to the AnnotationTab, the start and end time is auto-filled-in based on the start and end of the anomaly.



3 The most important piece of information is a concise title...

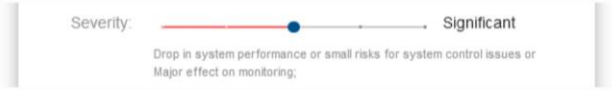
....and a good description of the event.

We simply try to summarize all information we currently have about the event.



4 Next we use the slider to insert a severity

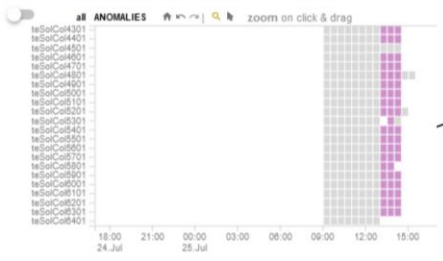
In our case the collector stagnation event is quite severe as it impacts the performance of the system. So we choose "significant".



5 And we can use the autocompletion menu to tag all affected datapoints.

Each datapoint in this list will be marked as "abnormal" when saving this event. All corresponding anomalies (which are raised within that timeframe for the tagged sensors) will be marked as "auto-confirmed".

We insert all sensors we previously identified in Q5 - that is, multiple collector temperature sensor.



6 All that is left is clicking on save.
With this, the event is annotated successfully.

T4 Diagnosis

We have annotated a new event. However, there are still many open questions.

As Diagnosis is a very explorative task, one does not need to solve all of the questions in this order.

Q13: Has this happend before? Is this a latent fault?

Q14: When did the event occur? is it still active?

Q15: Which sensors are affected by it?

Q16: How did the fault evolve? ← Lets start with this one.

Q17: What was the root cause of the fault?

Q18: Did the fault occur on sensor, component or system control level?

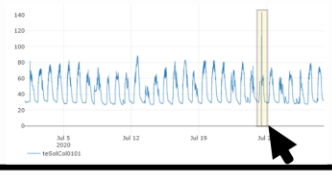
Q19: Is the fault related to other events?

Q20: What is the severity of the fault

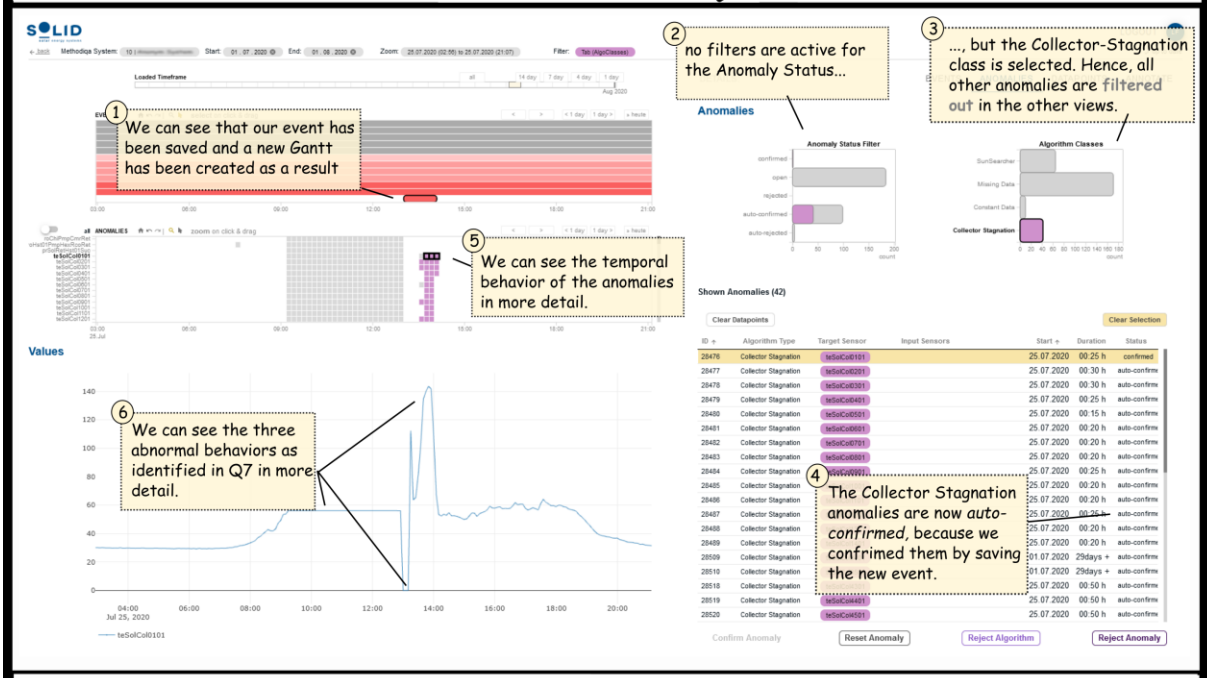
Q16: How did the fault evolve?

To gain more knowledge about the event, we look a little bit deeper and see if we can make more sense of the measurement values during the event (similar to Q6).

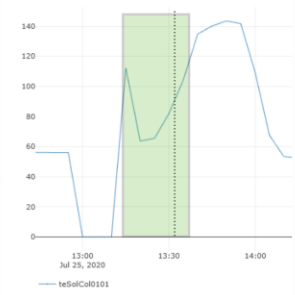
Our first move is thus to zoom in.



Now the screen looks like this.



Taking a closer look at the sensor values, one might wonder that everything looks fine from 13:15 to 13:45 o'clock.

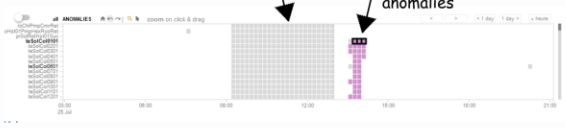


Similarily, the AnomalyHeatmap also shows no anomalies for this sensor at that time.

gap where no anomalies occured (corresponding to green area on the left plot)

"Missing data" anomalies

"Collector Stagnation" anomalies



Hence, either the events are not connected to each other.

Event A

Event B

no connection

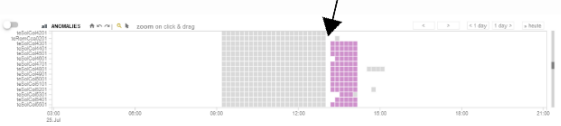
Or there is an "undiscovered" correlation between both of them.

Event A causes Event B

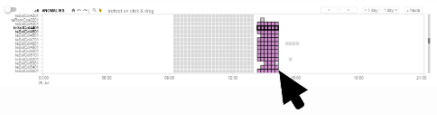
Event A causes Event C

And indeed, when scrolling down the AnomalyHeatmap, some collector temperature sensors show different anomaly-characteristics.

much smaller gap!

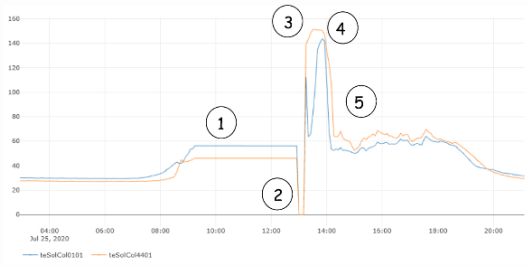


To analyse this, we select the anomalies by brushing



And select the sensors in the AnomalyTable to display the sensor data.

ID	Algorithm Type	Target Sensor
28518	Collector Stagnation	teSolCol4301
28519	Collector Stagnation	teSolCol4401



The results gain some interesting insights - the behavior is different.

1. Originally, both collector rows show similar behavior - both measurements get stuck at around 9:00 o'clock until 13:00 o'clock.

2. And both drop to 0°C at 13:00 o'clock.

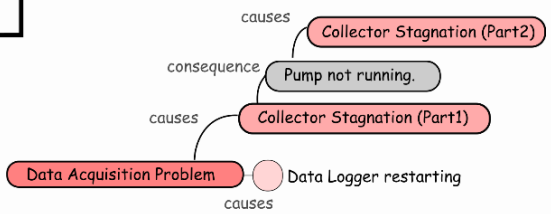
3. However, after the sensor readings are plausible again, the newly selected collector row (orange) shows high temperatures immediately ...

4. ... while the other collector (blue) goes into stagnation later that day.

5. And finally, the stagnation is resolved at around 13:30 o'clock.

This is an important piece of information for a domain user!

They may immideately form the following hypothesis:



1. **Data Acquisition Problem**
Some issue prevents the sensor data from being read. The system control may need to work with very limited amount of data.

2. **Data Logger restarting**
The sensor readings are finally back, but because of the Data-Logging starting up, some values are logged incorrectly.

3. **Collector Stagnation (Part1)**
Due to missing data the system control could not work properly and the collector subfield reached high temperatures.

Pump not running
As a direct consequence of the stagnation, the solar pump shuts down. This is to prevent damage due to vapor shocks and is a correct behavior of the system.

4. **Collector Stagnation (Part2)**
Because the pump is not running, the other collector subfield also heats up.

5. The issue is finally resolved as collectors cool down with less irradiation at the end of the day as the sun sets.

Let's find more evidence to this hypothesis.

One thing we can check is whether the pumps indeed are inactive during (3) to (4) and caused the stagnation of the second subfield

We click on the DatapointTab and select the corresponding sensor (Solar Pump Rotation) using the autocompletion menu

And indeed, the pump is not running while the stagnation is active.

With this we gained more knowledge, allowing us to answer some of the Diagnosis-Questions (at least in parts):

Q14: When did the event occur? is it still active?

The event happened on July 25, 2020.
It is not active any more (as the data logging is restored, the collectors cooled down, and subsequent days show normal behavior)

Q15: Which sensors are affected by it?

All sensors during the "Missing Data" event. Collector temperatures and pump signals during the "Collector Stagnation" event.

Q16: How did the fault evolve?

See hypothesis explanation above:
Missing Data ->
Data Logging Issue ->
Collector Stagnation (part1) ->
Collector Stagnation (part2)

Q18: Did the fault occur on sensor, component or system control level?

Failure of data acquisition.

Q19: Is the fault related to other events?

At least related to the "Missing Data" Anomalies
See Q16.

Q20: What is the severity of the fault

Reduced solar yield for this day as pumps were not running.
Small risk of component damage as stagnation occured.
=> Significant Severity (but far from severe or catastrophic)

While only few questions remain unsolved:

Q13: Has this happend before? Is this a latent fault?

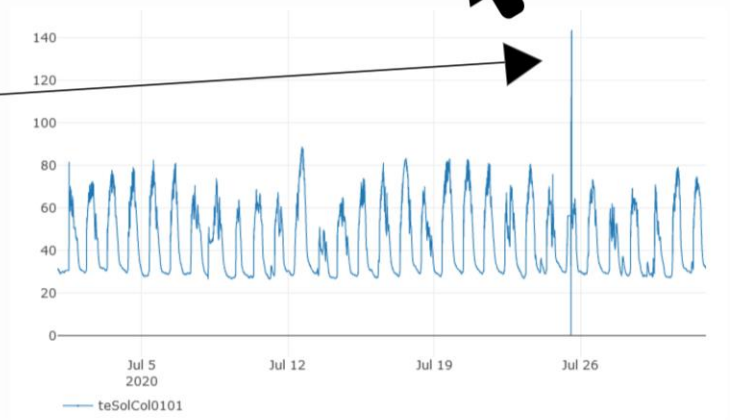
Q17: What was the root cause of the fault?

Q13: Has this happend before? Is this a latent fault?

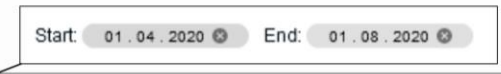
To answer this question, we zoom out and check a larger period of time.

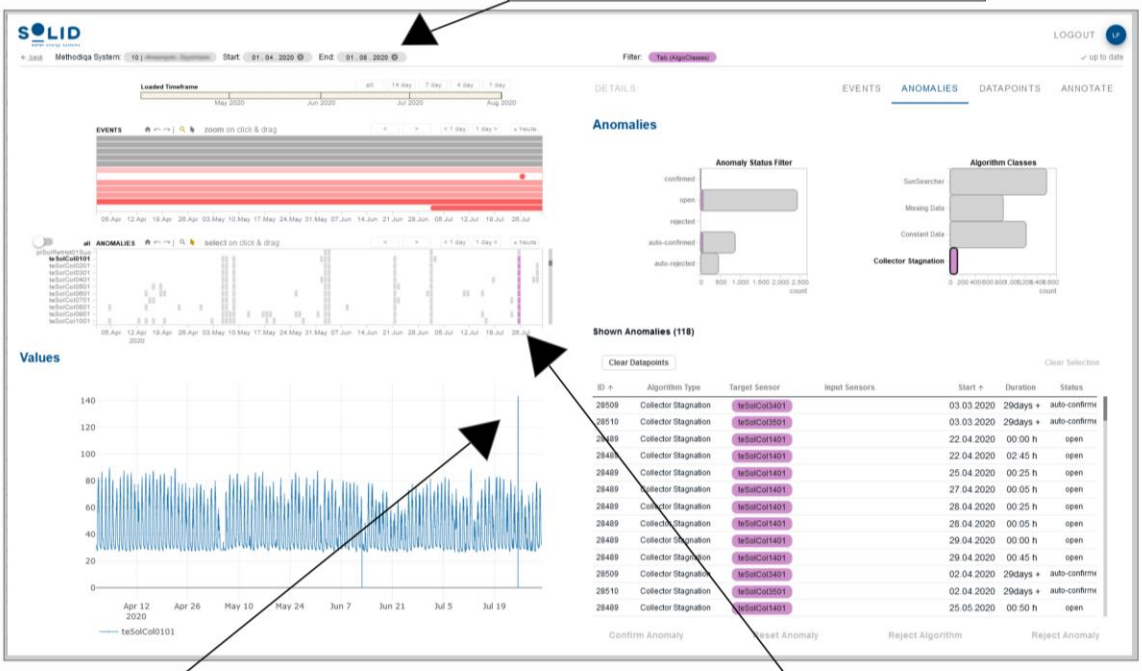
Apart from the event we just detected...

.... no other similar events are visible.



And the same is true when displaying 4 months of data (by specifying a different timeframe in the Header)





only event with such high collector temperatures.

only time where the Collector Stagnation Checker found anomalies

Thus we conclude that no similar event has happened recently.

Q13: Has this happend before? Is this a latent fault?

no

Q17: What was the root cause of the fault?

Unfortunately, the root cause could not be found using SunScreen alone. However it provides enough information for the system-operator to trace down the fault.

Q17: What was the root cause of the fault?

Analysing the equipment and talking to service personnel on site, it was found out that the data acquisition problems were caused by a severe thunderstorm

T3 Manual Fault Detection

Up until now, we have followed the workflow once from T1 Overview to T4 Diagnosis.

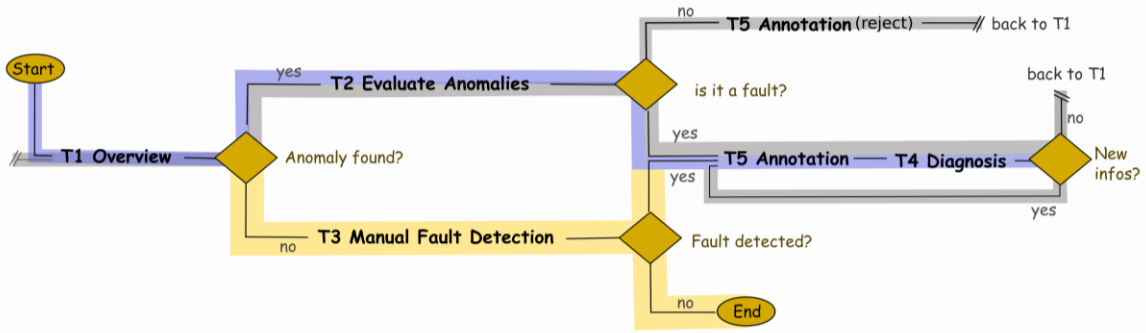
done above

In the next steps, the monitoring personell would annotate the new information from T4 and switch to analysing other anomalies (T1).

next steps

But eventually, all anomalies will be either rejected or confirmed. What is left is checking if there are any faults the anomaly algorithms missed.

final steps



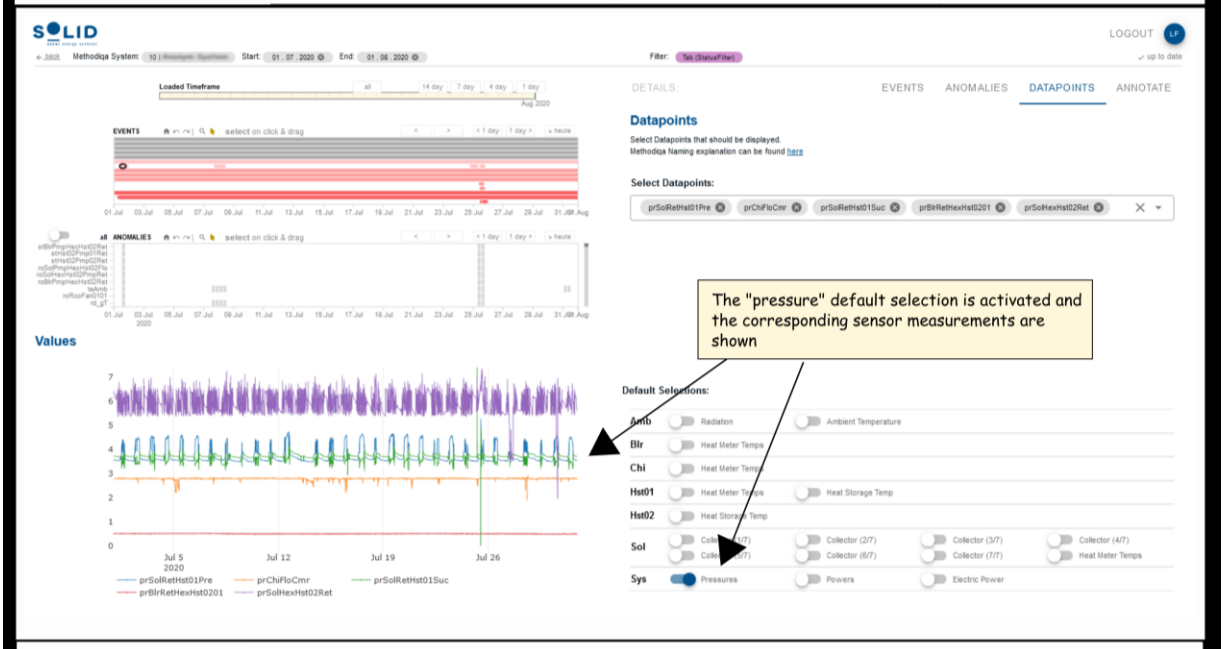
Q12: Are there any undetected faults?

To support this, SunScreen provides "default" sensor selections.

That is, sets of sensors defined by the domain-user beforehand, whose measurements hold important information to them.

For example the pressure sensors data could be viewed together, to detect any leakages in the pipes.

In case of a leakage the pressure would either slowly or drastically decline.



The "pressure" default selection is activated and the corresponding sensor measurements are shown

Fortunately, there seems to be no leakage.

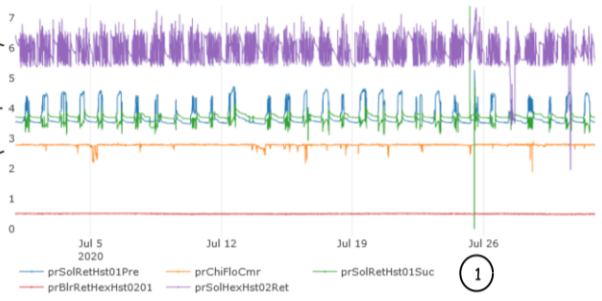
(1) However, the "Data-Logger restarting" Event also affects the pressure sensors.

Pressure in Heating Circuit: Typical variation based on pumps starting and stopping

Pressure in Solar Circuit typical daily patterns when pumps switch on and off.

Pressure in Consumer Circuit: mostly constant - the pressure drops are typical behavior.

Pressure in Boiler Circuit: constant as no boiler is used.



This way, the monitoring personnel would look through the other default selections as well, to check if the sensor values show any undetected anomalies.

If none are found, the monitoring session is completed.

Q12: Are there any undetected faults?

no

8 Discussion

This section reflects on the feedback of the domain users and the project as a whole. The first part of this section discusses potential improvements for SunScreen, while the second deals with encountered pitfalls and lessons learned during the project. Finally, the last subsection reflects on using data comics for presenting the design of *SunScreen*.

8.1 Future work

Although some issues were identified during the evaluation (see section 7), the stakeholders at SOLID are dedicated to further support the development of *SunScreen*, which shows the tool's potential. They argued that no other visual framework can yet combine sensor data, events, and anomalies. However, the open issues reported by the users also show that further development is needed. This section hence lists improvements that might increase the usability or functionality of *SunScreen*. Most ideas are derived from the evaluation feedback or were directly suggested by the users during the project's design stages.

The following increments might improve *SunScreen* without adding completely new functionalities:

Use Sensor-Position Data.
The sensor positions are not yet visualized in the current prototype of *SunScreen*. However, they are very valuable for the monitoring personnel. As written in the task and data abstraction in section 4, the sensor positions allow the user to interpret the measurements more efficiently as it contains important clues about which part of the system influences other parts. In addition, it also allows to identify and communicate the location of a fault more easily. As a result, this feature was requested multiple times during rapid prototyping and evaluation. While not enough time was available to tackle this feature in the prototype, there are many potentials in combining anomaly algorithms and sensor data with the sensor position, as done by Zhou et al. [11], for example.

More Accurate Anomaly Algorithms.
The sheer number of detected anomalies was overwhelming for all usability test participants. Some very sensitive algorithms, which generated many false positives, were even discarded to make the interaction with *SunScreen* easier. Hence, developing even more accurate algorithms might reduce the users' overhead.

More Straightforward Anomaly Algorithms.
Another angle of attack would be to use algorithms that are easier to interpret. For instance, the collector stagnation type algorithms are much easier to understand and evaluate for the domain experts than results from *SunSearcher* algorithms. That is because the former provides a much more straightforward explanation of the fault, while *SunSearcher* uses a machine learning technique that is more difficult to interpret.

Improved Tools for Analysing Anomalies.
In contrast to providing better algorithms, one could instead try to improve *SunScreen'*s functionality to filter and handle bad anomaly results. For example, grouping anomalies based on their attributes and setting many anomalies to "confirmed" or "rejected" together could save valuable time for the user and reduce the negative impact on false positives. Another idea would be to add views to gain more insight into the algorithm's internal mechanics. For instance, rules could be explicitly displayed in the case of rule-based methods, or decision trees could be visualized in the case of random-forest regressors. This way, users can more easily comprehend why certain anomalies are raised.

Display Residuals and Predictions.
Some algorithms (e.g., *SunSearcher* algorithms*)* also provide a residual value, showing the offset between predicted and measured sensor values. This information is not yet used in *SunScreen* but would make evaluating faults easier. For example, the predictions could be displayed in the sensor-values view on demand. This way, users may better understand why a particular algorithm raised an anomaly and what measurement values were expected instead.

Better Guidance.
Participants of the usability test argued that they were lost at some point in their interaction with *SunScreen*. This was prevented in the use-case study by explaining the designated workflow of *SunScreen* at the beginning of the session. However, a more appropriate way to solve this issue would be to improve the guidance provided by *SunScreen* itself.
To do so, one could try a more iterative approach for development: By removing some of the functionality, the complexity of *SunScreen* will be reduced. This way, users may feel less overwhelmed, and more knowledge about the remaining features can be gathered more effectively. By reintroducing features again, one can iteratively add functionality back to *SunScreen* without overloading the target users.

Chain Events together.
Events are currently considered single entities with no relation to other events. However, as described in the data abstractions, this is not true, for example, if other faults cause faults. Other examples are latent faults, which may reappear based on the system's operating conditions. When annotating such events, it would be nice to include these relations, such that the information is also visible to other analysts and system managers.

Annotation Suggestions.
To make annotation (T5) faster, one could develop automatic annotation suggestions, for example, based on historical results. In case of latent faults, it would also be nice to adopt event descriptions automatically to save valuable time.

Fix open usability issues.
In addition, the overall usability of *SunScreen* could be improved by fixing the issues identified by the domain users during the usability test: For instance, highlighting selected anomalies and events in the Sensor-Values view, introducing a hover line that is synchronized for all charts in the Time-Panel, or allowing to multi-select anomalies to change their status more quickly are three simple examples which would help the user in interacting with *SunScreen*.

In contrast to the section above, the suggestion below might introduce completely new angles of attack for solving the FDD tasks:

Detecting Trends.
The current prototype of *SunScreen* displays recent data from the solar plants (1 month by default). For diagnosis, it would be great to compare current measurements with historical data dating back multiple years. One example would be investigating a gradual pressure drop in the system because of a small leakage. While this fault can be found using anomaly algorithms, it is currently hard to evaluate such long-term faults as the shown period is limited.

Learn from Historical Events.
Data from manually added events could be used to train algorithms that raise an anomaly if similar events happen again. This could be supported “on the fly” anytime a user annotates a new event – the data is analyzed, and the specific patterns are learned to identify a new similar event. This would support the transition from manual to automatic fault detection. In the same

way, it could be checked if a similar event has happened before by searching through historical data. For example, a similar approach has also been implemented in Timefuse [43] and Visplore [9, 44].

Adding Benchmarks for Better Overview:
Developing and using benchmarks (i.e., key performance indicators) would allow for assessing the system's status more easily. While their results cannot be used to isolate a fault and perform Diagnosis, they might help identify which anomalies are relevant and which do not correspond to any critical fault. On the other hand, the anomaly algorithms worked well in isolating affected sensors and diagnosing faults in detail but failed to provide a good overview of the system status. Hence, the strength of both approaches could be used by combining the information. Thus, benchmarks may be used to identify essential faults (T1), while automatically detected anomalies provide more insights about affected sensors and diagnosis leads (T2, T4).

Display Sensor-Correlations.
Machine Learning algorithms like *SunSearcher* use correlations between sensors to learn the behavior of specific sensor measurements. Showing the user this information might help them understand the algorithm better. In addition, the correlations might also come in handy during diagnosis (T4) to identify the true culprit of a fault. For example, a network graph could display the sensor correlations with links between correlated sensors. Anomalies might be encoded by coloring sensors and correlations accordingly. This way, a user might easier identify sensors that are responsible for many raised anomalies and trace the correlations to find the real culprit.

8.2 Encountered Pitfalls

This subsection reflects on the whole design study. It discusses pitfalls that were encountered during the project and compares them with the ones identified by Sedlmair et al. [15]:

No Data Available / No real task.
The anomaly detection algorithms were introduced only shortly before the development of *SunScreen* was started. Thus, the domain users were not yet fully accustomed to interacting with these new anomalies and algorithms. As a result, both the requirement analysis and the evaluation of *SunScreen* are less accurate. While the domain users knew what task they want to solve, it was impossible to watch the target user carrying out the Evaluate Anomaly task (T2) in person. In addition, feedback to *SunScreen* is mixed up with feedback to the anomaly detection algorithms like *SunSearcher*. In hindsight, a visual framework focusing on manual fault detection would have been easier to validate. Nevertheless, it was a strategic choice of the stakeholders to opt for automatic fault detection to benefit from the automatization. This pitfall is equivalent to the pitfall PF-4 identified by Sedlmair et al.

The topic is too large.
Another problem was encountered during the Winnow stage of the project: Initially, the project's scope was too big. As a result, many resources were spent on parts of the application that are not that important and finally not presented in this work. Asking questions like '*Is there enough time to tackle this topic?*' or '*Can the topic be reduced to a smaller but equally interesting one?*' would have helped to focus on the more essential aspects of the topic.

Bias because of too much domain knowledge.
The main author has worked in the solar thermal domain for multiple years. However, Seldmair et al. identified "learning their problems/language too much" as a common pitfall for

vis design studies (PF-18). Thus, task and data abstraction, the considered design space, and the evaluation of *SunScreen* might be biased. Further bias is introduced because the design stages were solely carried out with people from SOLID, while no other company was asked to provide feedback. Knowing that this pitfall exists, we tried to reduce the risk of bias by actively questioning the results of each stage and by comparing them with related work extensively. For instance, the data and task abstraction were intentionally compared to related work, checking for any similarities or differences that might have been introduced by bias. Similarly, the considered design space was broadened by deliberately adapting ideas from related work in early prototypes and challenging them with domain users. In addition, weekly meetings with a visualization expert not working in the domain provided another safety measure to reduce bias.

8.3 Reflections on Data Comics

Inspired by Bach et al. [14], we used Data-Comics to present *SunScreen*. We believe that doing so allows the reader to follow the interactions with the visual framework much easier. And indeed, the feedback from the reviewers suggests that this approach works very well. As Data-Comics have not yet been used in design-study papers, this section shares some of our experiences. Hence, the first two subsections analyze the benefits and drawbacks of Data-Comics, while the latter subsections deal with design choices and share our experiences in creating comics.

8.3.1 Comparison to Figures and Text.

This subsection analyses the benefits and drawbacks of using Data-Comics compared to the typical approach of using text and images separately. The following list is partly inspired by the reviewers' feedback but is primarily based on the author's personal opinion.

Overall, the Data-Comics received excellent feedback from the reviewers. In total, we asked three domain users, one visualization expert, and one non-domain user to review different comic variations (see reading order and panel contrast in the following subsections). In addition to the feedback on the variations, they all unprompted praised that the data comics work very well compared to the typical approach of using text and images separately. The following aspects might be a reason for the positive feedback:

Better reading flow. In a typical approach, authors of design studies mainly rely on text to describe their visual framework while providing screenshots of the application as figures for visual references. Hence, the reader will often switch between text and images and has considerable overhead to comprehend the users' interactions within the application. In contrast, Data-Comics allow the mixing of text and images such that passages that correlate to each other are closer together. Hence, eye movement distance is reduced, and it is easier to follow the flow of the narrative.

Easy to read (A picture is worth a thousand words). In addition, Data-Comics allow us to show interactions in much greater detail than what could be accomplished by describing them with words only. For instance, interactions like brushing can be shown easily using a Data-Comic by using panels next to each other depicting the interaction. In contrast, describing the same interaction with words might be either very verbose and difficult to follow or omit essential details from the user.

Better Guidance and Focus. Another benefit of Data-Comics is that it is easier to control the level of detail and guide the reader through the narrative.
For instance, images provide a large amount of detailed information that can be processed quickly without creating much overhead. On the other hand, text is often required to explain

the visuals in more detail. Thus, text can tell the main story, while images communicate additional details fast and effectively.
In addition, Data-Comics intuitively allow adapting the level of detail shown in each panel. For example, one can zoom in on an important subsection of the visual framework, omitting sections that are irrelevant to the current narrative. In the same way, sketches can be used instead of screenshots of the application to lower the level of detail shown to the reader. Thus, the readers' attention is focused on what the comic designer intends to show.

However, there are also some disadvantages of Data-Comics:

A considerable amount of space is required. One drawback of the Data-Comics is that they need considerable space. Our walkthrough consists of about 15 pages that cannot be compressed further. In contrast, the conventional way of showing text and images separately would have been much more space efficient. While the comic might still be easier and faster to read, this is a limiting factor for publications, as journals often limit the maximum number of pages.

Less suitable for Keyword-Lookups. Another drawback reported by a reviewer is that searching for specific keywords does not work on data comics. That is because we included the comics as images. However, this may be resolved using different software that allows embedding the comics without conversion to images.

Editing takes more time. Finally, one disadvantage compared to the traditional method is that revising Data-Comic requires more time. Especially if the panel size is adapted, this might also require rearranging the subsequent panels. For example, changing the font size or adding one panel at the start of the comic may result in re-drawing the whole comic again. In addition, the comic must be converted to an image and included in the document after each change. Hence, the revision process of Data-Comics is much more involved than editing text or images using the traditional approach of text and figures.

8.3.2 Comparison to Video

An alternative method to present visual frameworks is providing additional video material or demos to try out the prototypes interactively. Compared to Data-Comics, this allows the user to get an even deeper understanding of how the visual framework can be used. In addition, the showcasing is more transparent than Data-Comics, as each step is shown in full detail. However, the main drawback of videos and demos is that they cannot be printed and hence are more difficult to include in publications. Another disadvantage of videos might be that it is hard to perceive them in detail and get the right pacing for every viewer [45]. In contrast, Data-Comics allow each reader to choose their reading speed individually.

8.3.3 Creating Data-Comics

This subsection shares our experience on how we created the data comics. Hence it explains our workflow and which issues and bottlenecks we experienced during creation.

The comics were created using the following workflow:

1. **Storyboard:**
   First, we created a storyboard of the comic using a whiteboard and pen and paper. At this step, the main goal was to determine a rough storyline without focusing too much on details or how the result may be arranged into panels. For the Overview (section 6.2), we decided to give a short introduction to *SunScreen* by subsequently describing each part of the visual framework. In contrast, the Use-Case Scenario (see section 7.3) follows a more chaotic path, focusing on

how a user might interact with the visual framework.

However, one reviewer argued that we focused too much on explaining screenshots than conveying a story as typical comics do. He felt that our Data-Comics had too much information packed into each panel, making it harder to read. This might have resulted from choosing a rather factual narration for our Data-Comic. Instead, following the guides from Bach et al. [14] more closely might have helped to make our Data-Comics more engaging. Hence, better results might have been obtained if presenting only one message per panel or using a more person-centered storyline.

2. **Plausibility Check:**

   After finishing the first draft of the storyboard, we tried to re-enact the story using the implemented visual framework. Reacting with actual data helped us to ensure that no critical step was missed and that interactions were authentic. In case of deviations, the storyline was updated – often leading to the whole story being rewritten. We also took screenshots of each user interaction so they could be used for drawing the comic at later steps if needed.

3. **Drawing:**

   Next, the comic was drawn using Inkscape [46] using an A4 page layout. Using the screenshots from before, the main content of the comic was drawn following the storyline. However, not every interaction is shown in full detail. Instead, unimportant interactions were pruned to keep the length of the comic to a minimum. In addition, hand-drawn sketches were inserted to describe some aspects in a more abstract level of detail.

   During the drawing stage, we found that the primary focus should be on creating a concise story, while less attention should be put on the layout of the panels. Instead, we first used one large panel spanning the whole width and created individual panels only after the content of the sections (about half of an A4 page) had been finalized.

   It is helpful to check that the font size is large enough to read the content well (we expect >9px to work well). Increasing the font size later might require rearranging many panels, creating considerable effort.

4. **Editing:**

   Next, the whole comic was spellchecked, and panels and contents were rearranged. As a final step, the comic panels are exported as .png files and inserted into the document. However, in our case, the comics are often larger than one page and must be cut to fit on the page. Unfortunately, locating sections where cuts are possible and meaningful is not always easy. To compensate for this, we tried to keep the height of the panels as narrow as possible, so cuts could be made more flexible (see Figure 25).

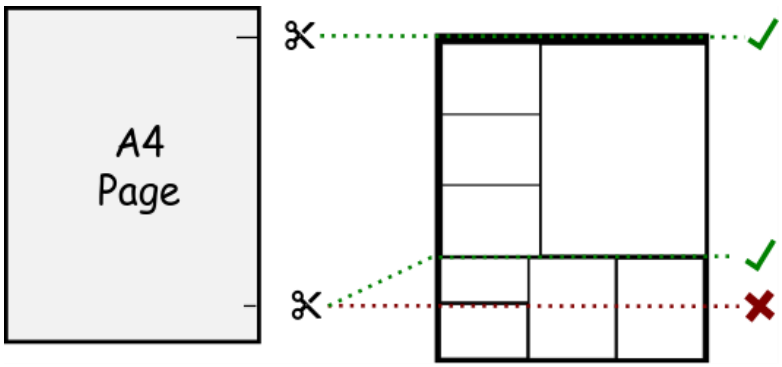


Figure 25: Comics need to be cut to fit the document pages. However, cuts cannot be made at any position (red dashed line). Hence, try to keep the panels' height down to get more potential cutting spots (green dashed line).

In our case, about 30 hours were spent creating the comic for the walkthrough, leading to an average creation speed of 2 hours per page. Most reviewers suspected much more work would be needed to create the data comic. However, creating the comic felt intuitive as one could choose between using text, screenshots, or sketches on demand. This flexibility allows presenting the application just like one would explain it to a new target user in a field study. In addition, details of the visual framework can be shown using screenshots instead of spending too much time explaining them. This saves time and makes the creation of Data-Comics more enjoyable compared to the text-and-image-separately alternative.

However, we also found that editing the Data-Comic can be very tedious. Aligning the comic on pages requires much time as cuts must be made manually. In addition, more severe changes during editing might require the whole comic to be rearranged again. For example, rearranging the panels of the data comic (due to increasing the font size) took about 1 hour per page.

8.3.4 Comic Design

This subsection tries to justify some of our design choices for Data-Comics, as some questions regarding layout and design arose while creating the comic. Hence, multiple different layouts for the comics were shown to the reviewers to see which worked best for the specific task at hand. Each layout was reviewed by one visualization expert, one non-domain user, and up to three domain users. Interestingly, the feedback to the comics considerably differed when showing sketches compared to comics with actual content describing *SunScreen*. This emphasizes that the content of the panels influences which comic design works best. Tests hence should be made using actual content and focus on readability instead of aesthetics.

Panel Design. Comics consist of multiple panels – each displaying a part of the story the comic conveys. To successfully guide the reader through the story, panels must be distinguishable from one another without distracting the reader.

Conventional comics use a white background with narrow black borders around the panels, while images fill up the entire content of the panel. Due to the high contrast to the background, it is easy to focus on the content and distinguish the panels. However, in our case, the content of the panels is mainly white. As a result, the high contrast of the black borders to the remaining comic feels distracting, making it harder to distinguish the content from the background. To compensate, we switched to a black background to provide better contrast between the comic and the background, even though this creates some visual clutter compared to conventional comics (see Figure 26).

In hindsight, however, we suggest working with gray borders or panels to decrease the contrast between text and panels but still keep the panels distinguishable (see the bottom part of Figure 26). The feedback from the reviewers suggests that the best choice for the panel design depends on the color of the visual framework to present. In case many colors are used, “grey panels” or “gray background” might work well as no additional colors are introduced that may conflict with the colors of the framework. On the other hand, “colored text” seems to work best for a less colorful tool like *SunScreen*. However, the design of the Data-Comic used within this work could unfortunately not be changed anymore due to time limitations.

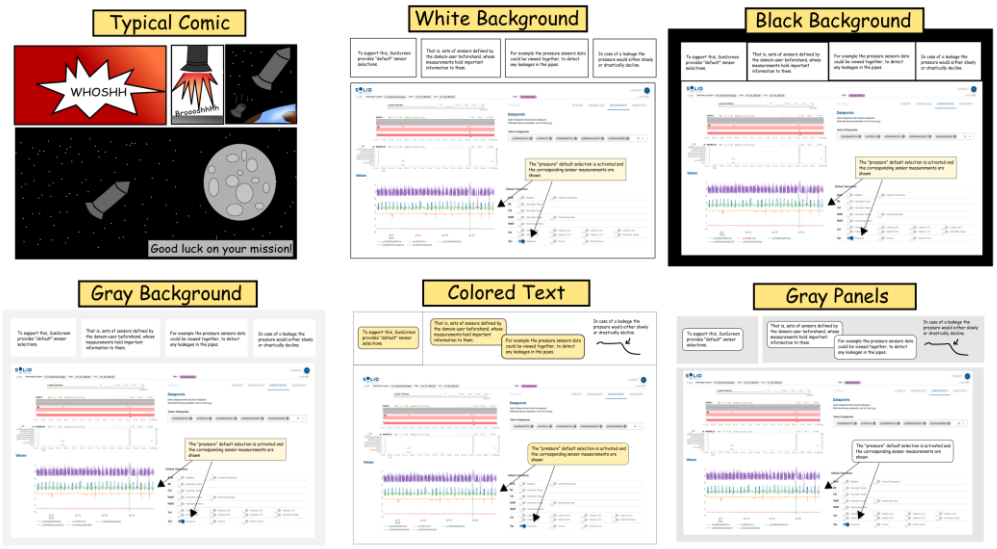


Figure 26: Comics presenting SunScreen using different styles compared to a typical comic (top left) (drawn by the author). The white background (top middle) looks very tidy at first glance. However, the black borders might distract from reading the content due to the contrast. The black background (top right) looks more cluttered but allows the reader to focus on the text and images more, in our opinion. Using gray colors (bottom row) for the background or borders decreases the contrast while keeping the panels distinguishable. Colored text bubbles (bottom middle) can be used to put even more focus on the text.

Headers and Sections. Two of our reviewers suggested using colors for the headers of each section and subsection of the comic. They argued that this would allow them to keep an overview of the story more easily. We updated the data comic accordingly and found this to work very well (see inner right of Figure 27). One alternative idea was to color the background of the panels instead, for example, by alternating white and gray panel backgrounds or using different colors for every section individually. However, this resulted in inferences with the contents and did not allow them to distinguish between main and sub-sections. Finally, the last considered option was full-width headers for each section (see outer right of Figure 27). The advantage of this option is that it is easier to see the section changes. However, it also reduces the space available for the panels. Thus, we opted for the "Header Color (above Panel)" option for this work.

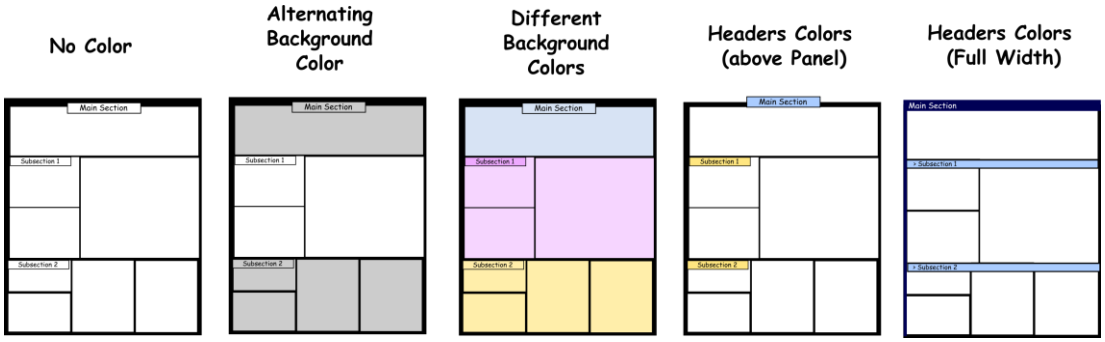


Figure 27: Variations for encoding section changes at Data-Comics. It is hard to distinguish sections using no colors (outer left). Alternating the panel background (inner left) may perceptually lift some panels in the foreground and might clash with the content of the panels. The same applies to using different background colors for the panels (inner right). Using different hues might indicate a connotation of each section and lead to higher inferences with the contents. Coloring the headers (inner and outer right) received the best feedback. Finally, we chose the "above Panel" option to communicate section and sub-section changes.

Facilitating Reading order. Some reviewers argued that the reading order for the panels was unclear for some parts of the comic. Most of them applied the western convention of reading panels from left to right and top to bottom. However, they struggled when panels of different sizes were placed next to each other. One example can be seen in Figure 28, with multiple small panels on the left and one large panel on the right. We tried to make the reading order clear by modifying the panel borders. To do so, we challenged multiple alternatives with the reviewers (one visualization expert and two domain users).
We finally opted for the "Small Gaps" as it provides the best ratio of high reading guidance to low visual clutter (see Figure 28). However, in hindsight, the "Text Bubbles" variant would also have been a good option. In contrast to the other options, it allows overlapping bubbles or extending them outside the panels to save space. It also received good reviewer feedback concerning facilitating reading order, similar to the "Small Gaps" alternative. Unfortunately, this option was discovered late in the Data-Comic design and hence was not used in the final work due to limited time resources.

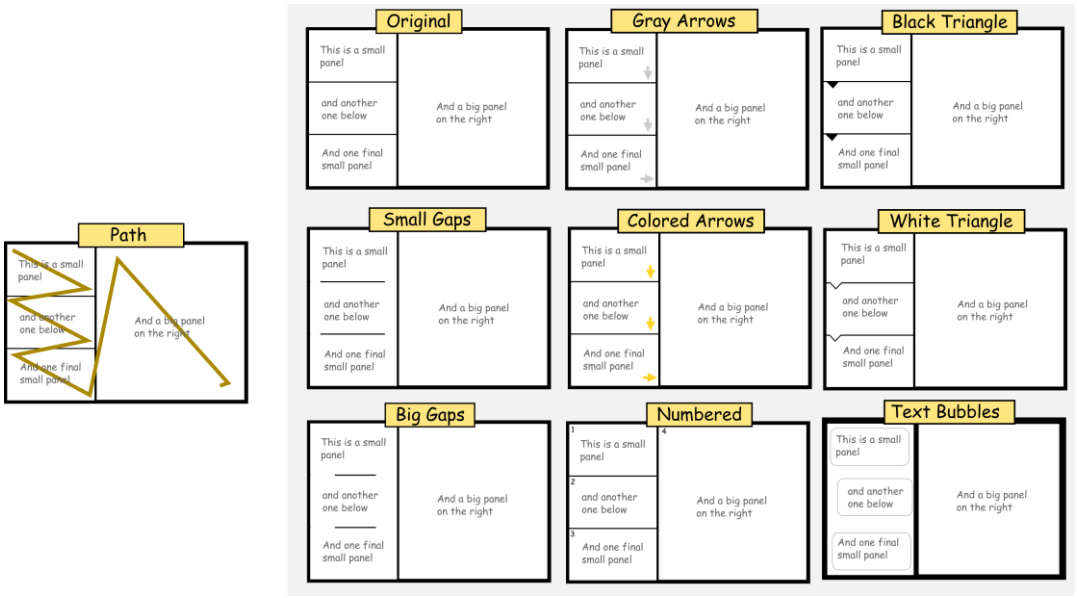


Figure 28: Comics with modified panel borders to facilitate the reading order. The reading order of the "original" version was unclear to the reviewers. The comics using arrows and numbers did interfere too much with the content of the panels. The comics using the path, triangles, and big gaps created too much visual clutter, distracting the user from the text. Finally, the small gaps were used for the comic, as they effectively guide the reader through the panels while not distracting too much. In addition, text bubbles are another good option to facilitate the reading order, limit space requirements, and create a more comic-like appearance.

9 Conclusion

This work studied visual Fault Detection and Diagnosis for solar thermal systems. More specifically, we introduced *SunScreen*, a visual framework supporting the monitoring personnel, and also provided a thorough requirement analysis working closely together with domain users from *SOLID Solar Energy Systems GmbH*.

Our investigation showed that three datasets are needed to support the monitoring personnel during their work. First, sensor data provides essential information about the system's performance. Second, system events provide contextual information about what happened in the system. And finally, results from anomaly detection algorithms hint at suspicious system behavior. Unfortunately, no visual framework that combines all these datasets exists, as system event data is rarely supported by any tool.
The interviews also revealed five high-level tasks that need to be performed by the target users. Most

of the tasks (gaining an overview, evaluating anomalies, performing manual fault detection, and performing diagnosis) were also identified in related work. However, we especially found “annotating anomalies” to be an essential task for the monitoring personnel, which is rarely emphasized by related work.

SunScreen is the first visual framework that supports all five tasks and combines the information of all three datasets. It allows the users to analyze the complex multi-variate system data, evaluate automatically detected anomalies, and view system events which give vital hints to interpret the data. This way, the user is provided with all the required information to complete their tasks. By annotating discovered events, *SunScreen* allows saving the information for future sessions. Thus, more and more knowledge about the system can be gathered using this tool, automatically sharing the results with fellow analysts.

However, the evaluation showed that *SunScreen* is still a work in progress. Even after improving the algorithms and drastically reducing the number of false positives, a vast number of anomalies remained. With most of them corresponding to minor faults or outliers only, it is too overwhelming and tiresome to inspect all of them in detail. This indicates that the goal should not be to find all anomalies but only to find the relevant ones instead. Hence, future work could focus on deriving more precise algorithms or finding ways to filter-out unimportant anomalies.
Other concerns indicate that *SunScreen* needs more guidance in general. Having identified some possible improvements, we plan to continue our work with SOLID and further evaluate and refine the visual framework. Nevertheless, the results of *SunScreen* are already promising. The walkthrough clearly shows evidence that *SunScreen* is usable. At the same time, the feedback from the participants emphasizes that integrating system events and supporting annotation is an integral first step to improving the quality of monitoring.

Finally, this work also contains a first example of using Data-Comics to present visual frameworks. Even though some issues remained, the feedback indicates their success. Compared to typical approaches, the Data Comics seem especially well suited to guide the reader through the design of visual frameworks and explain their interactions in detail. We believe this medium has great potential and should be further studied and applied in future Design Studies.

10 References

[1] Schneid Gesellschaft m.b.H. , "Winmiocs70," Schneid Gesellschaft m.b.H. , [Online]. Available: https://schneid.at/visualisierung/. [Accessed 13 November 2022].

[2] Siemens Aktiengesellschaft, "SIMATIC WinCC V7," Siemens Aktiengesellschaft, [Online]. Available: https://new.siemens.com/de/de/produkte/automatisierung/industrie-software/automatisierungs-software/scada/simatic-wincc-v7.html. [Accessed 13 November 2022].

[3] Atvise, "Atvise Scada," Bachmann Visutec GmbH, [Online]. Available: https://www.atvise.com/de/produkte/atvise-scada. [Accessed 13 November 2022].

[4] T. Liu, B. Lee, S. Kandula and R. Mahajan, "NetClinic: Interactive Visualization to Enhance Automated Fault Diagnosis in Enterprise Networks," *IEEE Symposium on Visual Analytics Science and Technology,* pp. 131-138, 2010.

[5] L. Shi, Q. Liao, Y. He, R. Li, A. Striegel and Z. Su, "SAVE: Sensor anomaly visualization engine," *IEEE Conference on Visual Analytics Science and Technology (VAST),* pp. 201-210, 2011.

[6] M. Steiger, J. Bernhard, S. Mittelstädt, H. Lücke-Tieke, D. Keim, T. May and J. Kohlhammer, "Visual Analysis of Time-Series Similarities for Anomaly Detection in Sensor Networks," *Computer Graphics Forum,* vol. 33, pp. 401-410, 2014.

[7] M. Riverio, M. Lebram and M. Elmer, "Anomaly Detection for Road Traffic: A Visual Analytics Framework," *IEEE Transactions on Intelligent Transportation Systems,* vol. 18, no. 8, pp. 2260-2270, August 2017.

[8] C. Arbesser, F. Spechtenhauser, T. Muhlbacher and H. Piringer, "Visplause: Visual Data Quality Assessment of Many Time Series Using Plausibility Checks," *IEEE Transactions on Visualization and Computer Graphics,* no. Vol.23, pp. 641-650, Januar 2017.

[9] M. Vuckovic and J. Schmidt, "Visual Analytics Approach to Comprehensive Meteorological Time-Series Analysis," *Data,* vol. 5, no. 4, 2020.

[10] T. Gschwandtner and O. Erhart, "Know Your Enemy: Identifying Quality Problems of Time Series Data," *IEEE Pacific Visualization Symposium (PacificVis),* pp. 205-214, 2018.

[11] F. Zhou, X. Lin, X. Luo, Y. Zhao, Y. Chen, N. Chen and W. Gui, "Visually enhanced situation awareness for complex manufacturing facility monitoring in smart factories," *Journal of Visual Languages & Computing,* vol. 44, pp. 58-69, February 2018.

[12] P. Eichmann, F. Solleza, N. Tatbul and S. Zdonik, "Visual Exploration of Time Series Anomalies with Metro-Viz," *Proceedings of the 2019 International Conference on Management of Data,* pp. 1901-1904, 25 June 2019.

[13] H. Janetzko, F. Stoffel, S. Mittelstädt and D. A. Keim, "Anomaly detection for visual analytics of power consumption data," *Computers & Graphics,* vol. 38, pp. 27-37, 2014.

[14] B. Bach, N. H. Riche, S. Carpendale and H. Pfister, "The Emerging Genre of Data Comics," *IEEE Computer Graphics and Applications,* pp. 6-13, May/June 2017.

[15] M. Sedlmair, M. Meyer and T. Munzner, "Design Study Methodology: Reflections from the Trenches and the Stacks," *IEEE Transactions on Visualization and Computer Graphics,* vol. 18, no. 12, pp. 2431-2440, December 2012.

[16] T. Munzner, Visualization Analysis & Design, Boca Raton : London : New York : CRC Press, 2014.

[17] J. Brooke, "SUS: A 'Quick and Dirty' Usability Scale," in *Usability Evaluation In Industry*, London, Taylor & Francis Ltd, 1996, pp. 189-195.

[18] J. Brooke, "SUS: A Retrospective," *Journal of Usability Studies,* vol. 8, no. 2, pp. 29-40, January 2013.

[19] A. Bangor, P. T. Kortum and J. T. Miller, "An Empirical Evaluation of the System Usability Scale," *International Journal of Human–Computer Interaction,* pp. 574-594, 30 July 2008.

[20] R. Isermann and P. Ballé, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control Engineering Practice,* vol. 5, no. 5, pp. 709-719, 1997.

[21] European Committee for Standarization, BS EN 13306:2017 Maintenance. Maintenance terminology, Brüssel: CEN - CENELEC Management Centre, 2017.

[22] V. Venkatasubramanian, R. Rengaswamy, K. Yin and S. N. Kavuri, "A review of process fault detection and diagnosis: Part I: Quantitative model-based methods," *Computers & Chemical Engineering,* vol. 27, no. 3, pp. 293-311, 22 April 2003.

[23] M. Huovinen and V. Hietanen, "Monitoring and diagnosis of large scale industrial systems," *IFAC Proceedings Volumes,* vol. 39, no. 13, pp. 831-836, 2006.

[24] L. Feierl and P. Luidolt, "B-D3. Automated Monitoring Solar Thermal Systems," 23 September 2020. [Online]. Available: https://task55.iea-shc.org/Data/Sites/1/publications/IEA-SHC-T55-B-D.3.2-FACT-SHEET-Automated-Monitoring.pdf. [Accessed 13 November 2022].

[25] S. Knabl, C. Fink and W. Wagner, "Empfehlungen für Monitoringkonzepte in solarthermischen Großanlagen," April 2012. [Online]. Available: https://nachhaltigwirtschaften.at/resources/iea_pdf/endbericht_201603_iea_shc_task45_anhang04_monitoring_konzept.pdf. [Accessed 13 November 2022].

[26] Smartblue AG, "Smartblue," Smartblue AG, [Online]. Available: https://www.smartblue.de/. [Accessed 13 November 2022].

[27] C. de Keizer, K. Vajen and U. Jordan, "Review of long-term fault detection approaches in solar thermal systems," *Solar Energy,* vol. 85, pp. 1430-1439, 28 March 2011.

[28] G. Faure, M. Vallée, C. Paulus and T. Tran-Quoc, "Reviewing the dysfunctions of large solar thermal system: a classification of sub-systems reliability," *ISES Conference Proceedings (EuroSun 2016),* October 2016.

[29] SunReports, "SunReports," SunReports, 2011. [Online]. Available: http://www.sunreports.com/. [Accessed 13 November 2022].

[30] Meteocontrol GmbH, "VCOM Monitoring," Meteocontrol GmbH, [Online]. Available: https://www.meteocontrol.com/photovoltaik-monitoring/produkte/vcom-cloud/vcom-monitoring. [Accessed 13 November 2022].

[31] SMA Solar Technology AG, "Sunny Portal Professional Package," SMA Solar Technology AG, [Online]. Available: https://www.sma.de/produkte/monitoring-control/sunny-portal.html#Professional-Package-108542. [Accessed 13 November 2022].

[32] Solar-Log GmbH, "Solar Log Web Enerest TM XL," Solar-Log GmbH, [Online]. Available: https://www.solar-log.com/. [Accessed 13 November 2022].

[33] Plexlog, "PlexLog Portal," PLEXLOG GmbH, [Online]. Available: https://www.plexlog.de/web/. [Accessed 13 November 2022].

[34] QOS Energy, "QOS Energy Quantum," QOS ENERGY, [Online]. Available: https://www.qosenergy.com/solution/renewable-plant-monitoring-analytics-om-reporting/. [Accessed 13 November 2022].

[35] 3E, "3E SynaptiQ," 3E SA, [Online]. Available: https://3e.eu/our-platform. [Accessed 13 November 2022].

[36] SOLYTIC GmbH, "Solytic," SOLYTIC GmbH, [Online]. Available: https://www.solytic.com/de/pv-monitoring/. [Accessed 13 November 2022].

[37] M. Bostock, V. Ogievetsky and J. Heer, "D³ Data-Driven Documents," *IEEE Transactions on Visualization and Computer Graphics,* vol. 17, no. 12, pp. 2301-2309, December 2011.

[38] Plotly Technologies Inc., "Collaborative data science," Plotly Technologies Inc., 2015. [Online]. Available: https://plot.ly. [Accessed 13 November 2022].

[39] P. McLachlan, T. Munzner, E. Koutsofios and S. North, "LiveRAC: Interactive Visual Exploration," *CHI,* 5-10 April 2008.

[40] W. Lidwell , K. Holden and J. Butler, Universal Principles of Design, Revised and Updated : 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach Through Design., Rockport Publishers, 2010.

[41] Y. Holtz, "Data to VIZ," [Online]. Available: https://www.data-to-viz.com/caveat/spaghetti.html. [Accessed 13 November 2022].

[42] S. Krug, Don't make me think! A Common Sense Approach to Web Usability, German language edition 2. Auflage ed., Heidelberg: Pearson Education Inc., 2006.

[43] KNOW-CENTER GmbH, "Timefuse," 2020. [Online]. Available: https://timefuse.io/. [Accessed 13 November 2022].

[44] Visplore GmbH, "Visplore," [Online]. Available: https://visplore.com/how-to-label-time-series-efficiently-and-boost-your-ai/. [Accessed 13 November 2022].

[45] B. Tversky, J. Bauer Morrison and M. Betrancourt, "Animation: can it facilitate?," *International Journal of Human-Computer Studies,* vol. 57, pp. 247-262, 2020.

[46] Inkscape Project, "Inkscape," 16 04 2020. [Online]. Available: https://inkscape.org. [Accessed 13 November 2022].