Computer Science and Information Systems 15(3):705–731 https://doi.org/10.2298/CSIS180126034H

The CRI-Model: A Domain-independent Taxonomy for Non-Conformance between Observed and Specified Behaviour ⋆

Christopher Haubeck,[1] Alexander Pokahr[2], Kim Reichert[3], Till Hohenberger[3] and Winfried Lamersdorf[1]

[1] University of Hamburg
Distributed Systems and Information Systems, Hamburg, Germany
haubeck, lamersdorf@informatik.uni-hamburg.de
[2] Helmut-Schmidt-University / University of the Bundeswehr Hamburg
Industrial Data Processing and Systems Analysis Group, Hamburg, Germany
pokahr@hsu-hh.de
[3] Adobe Systems Engineering GmbH, Hamburg, Germany
reichert, hohenber@adobe.com

Abstract Anomaly detection is the process of identifying *nonconforming* behaviour. Many approaches from machine learning to statistical methods exist to detect behaviour that deviate from its norm. These non-conformances of specifications can stem from failures in the system or undocumented changes of the system during its evolution. However, no generic solutions exist for classifying and identifying these non-conformances. In this paper, we present the CRI-Model (Cause, Reaction, Impact), which is a taxonomy based on a study of anomaly types in the literature, an analysis of system outages in major cloud companies and evolution scenarios which describe and implement changes in Cyber-Physical Production Systems.
The goal of the taxonomy is to be usable for different objectives like discover gaps in the detection process, determine components most often affected by a particular anomaly type or describe system evolution. While the dimensions of the taxonomy are fixed, the categories can be adapted to different domains. We show and validate the applicability of the taxonomy to distributed cloud systems using a large data set of anomaly reports and cyber-physical production systems by categorizing common changes of an evolution benchmarking plant.

Keywords: taxonomy of anomalies, anomaly detection, evolution, distributed cloud systems, cyber-physical system

1. Introduction

To develop the specification and to implement this specification in the system requires and create a high amount of knowledge from the developer and end user [10]. Anomalies are the non-conformances between these created specification and the actually observed behaviour. They occur as failures in the system due to not normal and unanticipated conditions and can lead to user dissatisfaction (e.g. decreased performance) [20], loss of data or

⋆ Extended version of the IDC 2017 conference paper “A Taxonomy of Anomalies in Distributed Cloud Systems: The CRI-Model” from Kim Reichert et al.

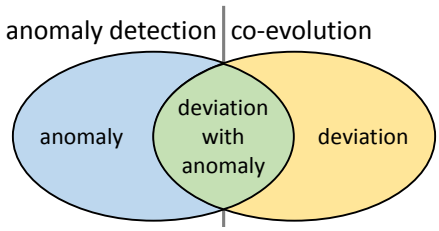

Figure 1. Relation of anomaly detection and co-evolution described as non-conformances

incursion of penalties due to a broken service level agreement. Until the reason for the anomaly is found and can be remedied, the system might be vulnerable to attacks or require additional resources. According to [23], e.g., Amazon lost roughly $25.000 per minute during a Thanksgiving weekend in 2001 due to a series of outages. Thus, early detection of system outages and automated recovery are very valuable for software companies.

But in a practical environment anomalies are not the only reason why nonconforming behaviour is detected during the operation of a software system, because nonconforming behaviour also results from incomplete or falsely specified behaviour. Then, the nonconforming behaviour is intended and just the detection process is not sufficient or up-to-date. These lack of specification between the documented behaviour and the operated system results from unplanned and unstructured changes due to time and cost pressure or staff turnover [27]and is for a long time well known as the loss of knowledge in the software community [22]. In this context, evolution describes the process of adapting software during its operation, in order to prevent it to degenerate relative to its environment. [10] To prevent degeneration detection of non-conformances between the specification and its systems becomes a major topic in the evolution research community [10]. This challenge is often coined as co-evolution of the system and its specification [29] and needs to be systematised and automated since the release times of changes are rapidly decreasing. For example, Amazon releases under the DevOps idea new or changed code of for their products every 11 seconds on average [2].

Both topics (evolution and anomaly detection) are generally isolated considered, but in this article we want to investigate the overlap between both topics by presenting a combined taxonomy. Therefore, we define *non-conformances* as differences between observable system behaviour and its normal system state that is documented in its specification. The resulting view is shown in figure 1. On the one hand, *anomalies* targeted in anomaly detection approaches exist that are undesired and malicious. These anomalies arise due to often temporal changes in the system condition and mostly have a malicious impact. On the other hand, *deviation* between the system behaviour and its specification can arise that can have a positive impact. These *deviations* often result from an adapted requirement and the software engineer reacts by changing the system to fulfil the new requirement without changing the specification. Anomalies and deviations are not disjunct as shown in figure 1. Such non-conforming nehaviour generally arise when the system is adapted resulting in a desired behaviour change (deviation), but the change also has an undesired side-effect that has not be foreseen (anomaly).

Following this broad definition of understanding non-conformances between system and specification, any system might exhibit a potentially large number of non-conformances.

In this regard, Ibidunmoye et al. state that a priori definition of all possible behaviours of an application is unrealistic [11, p.7]. However, every non-conformance, regardless of its type, will happen within a known system, which puts constraints on what and how behaviour within that system can occur. This also holds for evolution, because during evolution the current behaviour of the system is known and the system after evolution should act like the system before except the desired change. The assumption of this contribution is that while it may not be possible to predict all non-conformances that can occur, it might be feasible to provide an a priori classification for them. In their survey, Ibidunmoye et al. determine that *"a taxonomy of performance issues [. . .] will be highly essential for industry and academia"* [11, 4:24].

A taxonomy including *anomalies* enables developers to define malicious behaviour they want to detect, verify which types are already covered by detection or prevention policies and which are yet untreated. And also the detection of such anomalies benefits, because as Baddar et al. find *"[. . .], when the types of anomalies are not known a priori [. . .] selecting an fitting detection technique is not trivial."* [3, p.30]. Given the variety of anomaly types and manifold of metrics with an encompassing taxonomy this selection process of detection mechanisms and metrics can be supported by mapping different detection methods to different anomaly classes. At this point, a suitable taxonomy can serve as an analysis-tool by classifying types of already identified anomalies in a system and laying the groundwork of a mapping of approaches to detect future ones.

Further a taxonomy considering *deviations* allows to formulate underlying changes by specifying the cause of the change, the reaction as a deviation to the specification as well as its desired impact. Here, a taxonomy serves as a documentation of changes made to the system which is specified as a deviation detected in the system as a reaction to the triggering cause. This allows for specifying changes as differences to its previous version which is one of the most essential preconditions to understand and plan the evolution [29].

The here presented CRI-Model is a taxonomy which can be used regardless of the targeted type of non-conformances as well as the targeted domain. It provides general dimensions, but allowing domain-specific categories based on the system's characteristic by being adaptable at its lower category level.

This paper first gives an overview of related work in 2. In section 3 we explain our approach to designing the taxonomy. In 4 we present the resulting CRI-Model, and show the usage for anomalies within the domain of distributed cloud systems and deviations within the domain of cyber-physical production systems. In section 5 the applicability is shown by presenting an exemplary use of the CRI-Model in the different domains. Finally, in section 6 we discuss our takeaways and possible extensions of the taxonomy.

2. Related Work

Looking at the literature, in the domain of anomaly detection attempts have been made to create taxonomies of anomalies in (distributed) software systems. In [18], Mazel et al. define a taxonomy for anomalies in backbone network traffic, Plonka and colleagues introduce a taxonomy which focuses on network anomalies in general [24] and Mirkovic et al. present a classification of DoS-Attack anomalies [19]. Contrary to our point of view, Mazel's anomaly also contains normal events (since an important part of their taxonomy is the mapping of signatures, it makes sense to consider normal signatures for reasons of

exclusion as well) [18]. Furthermore, their result contains classes and subclasses of anomalies, while our model allows arbitrary combination of dimensional categories. Where they give a name to each distinct anomaly, we use the CRI-Model to describe it. In our opinion, this makes it far more likely to cover any type of anomaly that can occur. Especially if other domains are considered.

In their Attack-Taxonomy, Mirkovic et al. describe anomalies by intent and trigger only. Their dimensions can have subclasses, e.g. a classification by degree of automation has a subclass scanning strategy. Contrary to Mazel's taxonomy, their categories are not exclusive. Some anomalies can be sorted under different combinations of classes: e.g. Hitlist Scanning anomalies can be semi-automatic or automatic. The CRI-Model, on the other hand, does not consider intention, but therefore considers the trigger. While it can be helpful to understand an intentionally caused anomaly (e.g. for defence mechanisms), as a more generic attempt to describe anomalies, this dimension of intent seems too detailed for a generic classification usable for different objectives.

Torbergte et al. discuss two dimensions: the cause of a failure and the location of the component where it happens [28]. They determine eight categories of cause (e.g. operator error, node hardware, node software...) and four for location (front-end node, back-end node, network, unknown) where the anomaly first appears. They also differentiate between impact that leads to service failure (in our words impact on the upstream client) and impact that makes only singular components fail. Finally, like us, they distinguish an underlying flaw (the root causes in our case), which can become active in a certain system state, from the cause (our trigger). Just like us, they also create two-dimensional mappings of dimensions (cause-location) but mention their trouble to correctly map network-problems (mapping to network and 'unknown' as cause). In the CRI-Model of cloud systems, the reaction would be 'connectivity', and the trigger could be labelled as hardware change (since the network could be seen as part of the underlying hardware of a distributed software system) while the CRI-Model of cyber-physical production systems considers hardware in different disciplines. Contrary to the CRI-Model, their taxonomy might be most suited for mitigation strategies, not necessarily for anomaly detection, since its focus is on the impact.

In their survey paper on anomaly detection techniques, Chandola et al. do define different types of anomaly types (point anomalies, contextual anomalies and collective anomalies) [5]. However, these only take on the numeric aspect of an anomaly, and leave out the system and its components completely.

Avizienis et al.. finally, present the most detailed and simultaneously widespread taxonomy of anomalies we could find [1]. They distinguish three main components that are part of an anomaly, a *fault* which causes an *error* which can lead to a *failure*. Accordingly, they present a classification of faults and one of failures, the error itself, however, is not further distinguished. From the anomaly detection perspective, the monitoring data which the error (reaction) creates can be most essential. This lack of further distinction makes the taxonomy from Avizienis et al. unsuitable for a classification of anomalies with the purpose of automated detection and evolution. The CRI-Model, on the other hand, provides the reaction and component dimensions, which can help narrow down what kind and where from to collect data for detection.

In the domain of production system anomaly detection methods are published under the terms fault-detection and fault-diagnosis. A general overview is given in [12] and for

cyber-physical production systems specific in [21]. These approaches generally rely on models of the specific production domain and do not target a domain independent taxonomy. An example is the taxonomy given by Maier et al. in which anomalies are similar to our approach defined as differences between the observed system and a system model within a timing modelling formalisms like state machines, petri nets or markov chains [17]. However, the reaction category of the anomaly is generally not targeted in such approaches. At this point, our work can be compared to research that consider residuals to determine the type and location of the anomaly in production system like done by Roth et al. [26]. Such a residual approach could be used to determine the reaction and location of the CRI-Model. Nonetheless, the only indirect given taxonomies used by these approaches classify anomalies on a very low-level and do not combine fault types with more abstracted views like non-functional requirements as done in the impact dimension of the CRI-Model.

Further, in contrast to anomaly detection, this contribution also explicit considers co-evolution as deviations of the specified behaviour due to changes. Approaches like the one of Ladiges et al. explicit consider such deviations detected in domain-specific models [13]. The taxonomy of changes used by Ladiges et al., further described in [14] and extended by Vogel-Heusser et al. in [30, 31], serves as a basis for categories of the CRI-Model regarding deviations regarding Cyber-Physical Systems in this contribution. But in contrast the taxonomy of the CRI-Model focuses beside behaviour changes also on reactions and impacts. Therefore, they differ to the change taxonomies which focus on classifying changes with all artefacts involved in the development and evolution.

This difference holds also for general taxonomies of changes like the one of Buckley et al. in which similar to this contribution the *when*, *where*, *what* and *how* are used as a starting point [4]. Other examples are Fluri and Gall who focused on source code changes [9], Lehnert et al. who are considering especially the field of impact analysis [16] or Chapin et al. who are looking at more business related artefacts [6]. These taxonomies consider all kinds of changes and often focus on code changes or other artefacts like requirements and software tests. Instead, the CRI-Model focuses on the dependencies between the specified behaviour and the observable behaviour which can be monitored by considering (software) metrics. Nonetheless, our taxonomy should not be seen independent from change taxonomies, since the classification of the CRI-Model can be used to describe specific topics of a change taxonomy, e.g. the granularity and impact of changes. In [4] Buckley et al. these categories are for example just specified by small or severe and with the CRI-Model the granularity can be precisely described in form of resulting deviation.

From a methodical point of view the taxonomy of Ciraci et al. [7] for classifying software evolution is similar to our approach, because they also use a system model to specify a taxonomy for changes. Our approach also uses the idea of abstracted system models to allow domain-specific adaptation by defining categories based on the provided models of cloud systems and production systems.

3. Approach

To develop a taxonomy of non-conformances, we considered both perspectives of research and practice [25]. To narrow the wide range of literature and define the categories of the

dimensions of our taxonomy we focus for anomalies on distributed (cloud) systems and for deviations on Cyber-Physical Production Systems. In both areas, the authors of this article have extensive expertise in research as well as practise and we think that both areas cover the divergence of non-conformances rather well.

Our main focus is on anomaly detection. Thus, for anomaly detection we studied the literature in distributed (cloud) systems to investigate which types of malicious anomalies the different studies mentioned and detected. Papers were selected based on their focus on anomaly detection in distributed systems, cloud systems or micro-services (see appendix A). They had to contain some description of the anomalies, the data used for detection or any classification of anomalous system behaviour. We focused on how anomalies were described, what root causes were mentioned, what broke down or failed to function, what kind of effect was determined and which names were chosen to describe anomaly types. For example, some approaches differentiate between dimensions of root causes (*Intrusion Anomaly, Bottleneck Anomaly, Contention Anomaly, Flood-based Anomaly* and *Execution Anomaly)*, other approaches between the effect (*Performance Anomaly and Busy Loop Fault)*, or the location where they happen (*Network Anomaly, System Anomaly, Application-Anomaly, Physical Layer Anomaly, Utility Cloud Anomaly*). For the practice perspective, we explored reports on system outages as an example for anomalies from major online software companies. We analysed sixteen reported anomalies that happened in companies such as Google, Amazon or Yahoo (see appendix B). In almost all cases, these anomalies were full system outages (the impact needed to be intense enough to merit public reporting). Nonetheless, these reports added valuable input for the design of our taxonomy, particularly with regard to the different types of root causes (if they were reported), their manifoldness, and the possible failures they induced.

To extend the taxonomy to deviations due to changes we considered reports, literature and practical experience for a commonly accepted case study for cyber-physical production systems. We investigate common taxonomies of evolution as the one of Ladiges et al. [14] and Vogel-Heuser et al. [30, 31] as well as general change taxonomies of Buckley et al. [4] and Chapin et al. [6]. Therefore, we extent and refine our taxonomy to provide a combined taxonomy for non-conformances and investigate how evolution can be described with the taxonomy. To evaluate the combined taxonomy in practice we use implementations of evolution scenarios established in the domain literature that are described based on a real laboratory plant production of the Technical University Munich[4] that serves as a benchmark for evolution related research studies.

4. The CRI-Model

We condensed our findings from research and practice into a taxonomy, called CRI-Model (see figure 2). It captures *what* happened, *where* it happened and *when* it happened for one individual non-conformance of behaviour and specification. Regarding *when*, we identified three phases: *Cause*, *Reaction*, and *Impact*, accounting for the name (CRI). The *what* describes the types of cause, reaction or impact, whereas the *where* identifies the components or stakeholders, where the cause, reaction or impact occurred. Following this model, a non-conformance is described as one or multiple *causes* in or outside the system, which

[4] see https://www.ais.mw.tum.de/en/research/equipment/ppu/

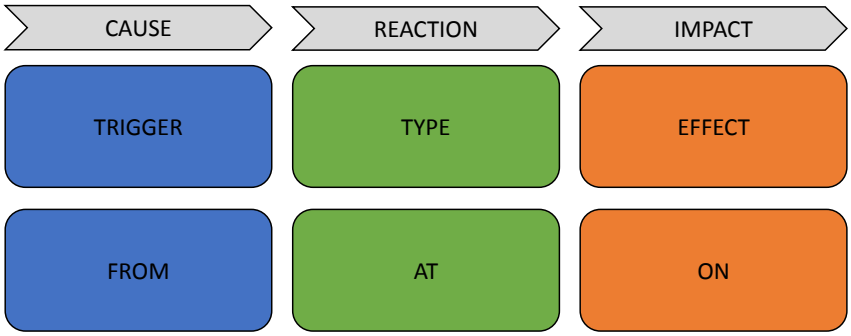

Figure 2. The domain-independent dimensions of the CRI-Model

trigger a chain of *reactions* at one (or multiple) components in the system, followed by one or multiple *impacts* of the system.

The resulting CRI-Model provides six *dimensions* for characterising an observed non-conformance. The first three dimensions express how the type of each of the three phases can be characterised. The dimensions express the *trigger* that initialised the cause, the *type* of the reaction that occurred and the *effect* which the impact has. Further for each phase one location dimension is specified which expresses where the cause is triggered *from*, *at* which component a reaction is performed and *on* which component the effect can be seen. The dimensions are domain independent, whereas additional domain-specific *categories* for each dimension must be added. The categories describe which concrete manifestations of each dimension are present in the specific system. Categories must be adapted to each system or at least system class and are shown for distributed cloud systems in section 4.1 and cyber-physical production systems in 4.2. In the following, first the six dimensions are described in more detail:

Cause Trigger is determined by the cause(s) that trigger the non-conformance. The cause must be the reason why the behaviour is triggered. Identifying and classifying the trigger of a observed behaviour often a necessary step in identifying the underlying problems because nonconforming behaviour might be determined using different information sources like metrics of system load, error-messages or text-logs.

Triggered From captures where the cause comes from (or who triggered it). It gives additional indications of how to handle an non-conformance. For instance, if nonconforming behaviour is triggered from outside of the system by a client call, it makes sense to apply detection on client-related metrics combined with metrics from components were the reaction usually happens. Also, it can be important to remedy the nonconforming behaviour or find a short-term fix as other parties might be necessary to solve the underlying problem. The location can further be used to classify non-conformances by sorting them regarding their trigger location.

Reaction Type describes how the system or engineer reacts to the cause. The reaction can have different characteristics. In case of a malicious anomaly this is often a rupture (often also named failure or error in the literature) of the system itself. The rupture is a reaction of the system due to a cause (deviation with anomaly) of e.g. the code and has normally a negative effect on the system. In case of an evolution the reaction can

also be beneficial by having a positive effect. For example, due to an change of the system logic a optimised behaviour is shown as a reaction. Such a reaction is desired. A Reaction(s) does not have to happen on the same component as the trigger(s). Because often, the trigger(s) set off a chain of events that eventually lead to a reaction(s) somewhere else in the system. For example in terms of Cyber-physical Production Systems a change in the production process of a processing machine might not be detected in the machine itself, instead the reaction is needed in the continued production process like the packaging of the product.

Reaction At distinguishes the types of components which are targeted by the reaction. This dimension is useful to identify bottlenecks, problems, and respective metrics on which the reaction occur.

Impact Effect is determined by the effect that the nonconforming behaviour has on the system itself or its stakeholders. This dimension allows distinguishing between the kind of effect by applying categories to the dimension, e.g., financial impact or reputation.

Impact On determines who or what is impacted.

When using the CRI-Model each of these dimensions can be filled repeatedly. Thereby, any observed non-conformance can sometimes be related to multiple categories or can affect multiple places at once, even if that should be avoided for a better categorization. For example, if multiple triggers are known which cumulatively require a reaction, they should all be listed under trigger. However, in many cases, only parts of the reaction chain are known. They can nonetheless be used to describe the non-conformance in the CRI-Model. For instance, in cloud systems an increased request count which triggers a rupture as its reaction at the load balancer is known, but the original root cause is a change in the public API, which provoked the users to request a service more often. Even if the original root cause is never detected, filing the increased request count as a trigger can still be useful to determine metrics for future detection or discuss resilience strategies.

4.1. CRI-Model of anomalies in Distributed Cloud Systems

The options to fill the dimensions are given by categories that are domain-specific. As a first domain for validation, we consider distributed cloud systems. Distributed cloud systems compute and store information geographically distributed over many locations in order to increase availability and reduce latency to the massive amount of users. To derive categories we modelled a generic distributed cloud system (see figure 3) based on a software system which provided the validation dataset of a big software company used in section 5. It consists of three types of components. The first components are workers that are in charge of executing the application logic. As the second components, flows transport information between workers. Persistence is the third part that implements any kind of data storage. The model has an upstream, which consist of cloud clients that execute calls on the cloud infrastructure. Further within a downstream other systems can serve as providers for external functionalities which are called by workers. The individual categories were derived from the literature and the practical reports.

The objective of this characterization of the CRI-Model are malicious anomalies (as non-conformances) that are observed within the back-end of a the highly distributed system. Following this objective the next section shows the derived categories for each dimension:

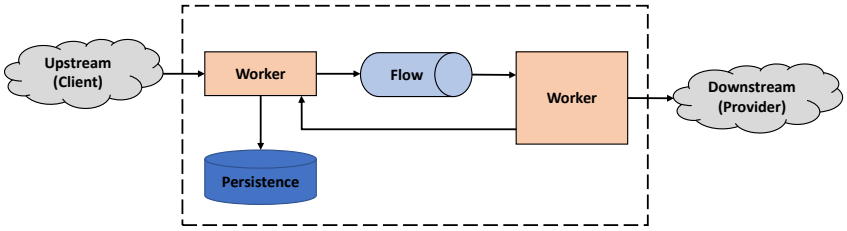

Figure 3. An abstract view on the different components of a distributed cloud system

Cause Trigger:

Software change: Any change which affects the logic of the application(s) to cause the observed anomaly. For instance, a developer deploying new code to the system or a downstream dependency changing its interface.

Hardware change: Changes regarding the underlying hardware of the system, including unplanned changes like corrupt disk writes, but also planned changes like scaling down a worker component.

Usage pattern: Any cause that is triggered by the amount or type of data flowing through the system, including, e.g., unusually large files, or a flash crowd event.

Triggered From:

Upstream: Any cause that is triggered from a client system or user of the cloud system's capability, e.g., intentional DoS-attacks, wrong usage of an interface (increased load, increased trigger of error-responses) or unnecessary retries from an impatient user.

Within: A cause triggered by a change in the cloud systems itself, including a changed worker configuration (e.g. scaling, expired keys), a crash of a persistence component, or an update to routing tables for one of the load-balancers controlling the flow in the system.

Downstream: Causes which are triggered by dependencies of the system, e.g., an outage of a cloud provider's queue system or a change to a public HTTP-API.

Reaction Type:[5]

Resource saturation: Any reaction caused by limitations of one or more resources, including lack of memory, disk space or CPU-bottlenecks.

Connectivity issue: A component becomes dysfunctional, because something else cannot be reached, e.g., a 3rd party dependency is not available (outage) or can't be accessed (wrong credentials).

Wrong logic: Dysfunctional behaviour of a component regarding the purpose of the application. This category fits if the flow through the system works just fine (connectivity and resource saturation), but there is a flaw in the handling of the data content, for example, due to a bad commit that was deployed.

[5] Since the CRI-Model focuses malicious anomalies only ruptures are considered as reactions.

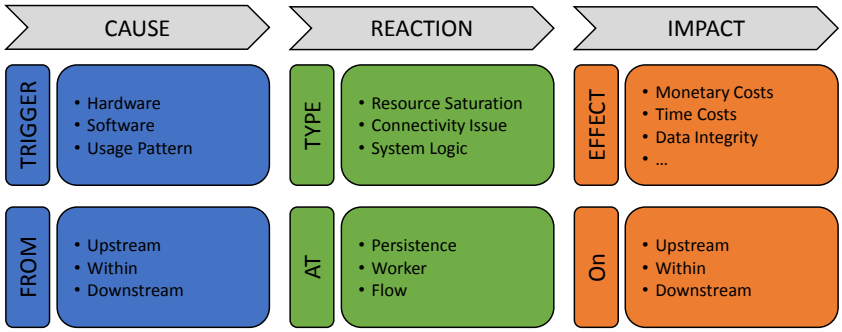

Figure 4. CRI-Model of a distributed cloud system

Reaction At:

Worker: Any affected component with application logic, which is hit by the anomaly, i.e. basically all components that transform data.

Flow: Components that deal with the flow of the system, forwarding, collecting requests or messages, such as load balancers, queues or event buses. Typical anomalies would be loss of events, bad throttling or overloading of queues.

Persistence: Any persistence component, like databases, system files or caches, typically concerning wrong storage of data, or bottleneck problems.

Impact Effect: [6]

Monetary costs: This includes anomalies which lead to broken SLA-agreements, or require an upscaling of resources, which costs more money.

Time costs: Often an anomaly leads to decreased response time, which can lead to unsatisfied users.

Data integrity: This category covers anomalies which trigger system states that make the system vulnerable to attacks.

Impact On:

Upstream: An anomaly can have negative impacts for upstream clients. Increased error-rate or loss of data that impact the client directly.

Distributed System: An anomaly can also negatively impact the distributed cloud systems itself like any decrease in the response time to clients.

Downstream: Any downstream provider of the cloud systems could be impacted, e.g., if an anomaly leads to an increase in requests sent to the downstream client.

The resulting CRI-Model with the categories is shown in figure 4. Regarding time all three phases cause, reaction and impact of anomalies are modelled based on the given generic distributed cloud model. Following the model, anomalies arising in a distributed

[6] The impact effect can be very broad and could, for example, include other QoS metrics or security aspects like vulnerability. But in this categorization we focus on the named three business relevant categories, because they can be derived from the investigated dataset of reports.

cloud system can be categorised by identifying the change that triggers in result of the system itself or its surrounding up- and downstream, describe the typical type of ruptures (reactions) that occur in one of the cloud elements and name the cost of the impact which is observed in or outside of the considered cloud.

4.2. CRI-Model of deviations in Cyber-Physical Production Systems

Production systems are typically long living and as a result undergo many changes and unforeseen incidents. Therefore, they are constantly modified and adapted which results in a continuous evolution process that is often, due to time and cost restrictions, not documented [14]. This lack of knowledge about the system results in deviations (as non-conformances) between the system's behaviour and its specification, because an undocumented change is generally not reflected in metrics used for the runtime detection and testing. This is why in these long-living production system detection of deviations between the system and its specification are of high interest [21]. These detection approaches do not assume faulty behaviour directly, but rather initialise a decision process to decide if an unexpected deviation is an anomaly or a deviation due to old specifications [13]. Therefore, detection of anomalies and deviations have a strong overlap which is why the following section shows how the CRI-Model can be applied also for detection of deviations.

Cyber-physical production systems are, due to there strong need for evolution, chosen as the domain. Cyber-physical production systems are characterized by interconnected physical and cyber components in the domain of production plants. Figure 5 shows the generic model we use for the categories of the CRI-Model in this domain. Production systems are characterized by processing the input (e.g. a physical raw product) with its mechanical parts that are connected by sensors and actuators. Both the mechanical as well as the automation hardware of the production systems is organized in plant components. Each component fulfils a specific capability of the production system. Both sensors and actuators exchange signals via a bus system. The system is controlled by a Programmable Logical Controller (PLC) which works in real-time and reads and writes binary signals of sensors and actuators. The behaviour is specified in software code that is normally written in a domain-specific language. Because the evolution of such interconnected systems is the objective of this categorization, beside the resulting output (e.g. a finished product) the system model also considers the context. The context is in terms of cyber-physical production systems mainly other connected interacting systems, the human operator and the implemented electrical, mechanical and software environment the production systems is situated in. In the following, categories based on this generic model designed for the manufacturing industry are presented:

Cause Trigger [7]*:*

Maintenance: Any deviation that is triggered by a change resulting from maintenance activities. These changes are done to maintain the original capability of the production system. This would, for example, be an exchange of an old sensor or a mechanical ramp because of wear and tear effects.

[7] The categories follow the view that maintenance is a cause or part of evolution (as, for example, presented in [27]).

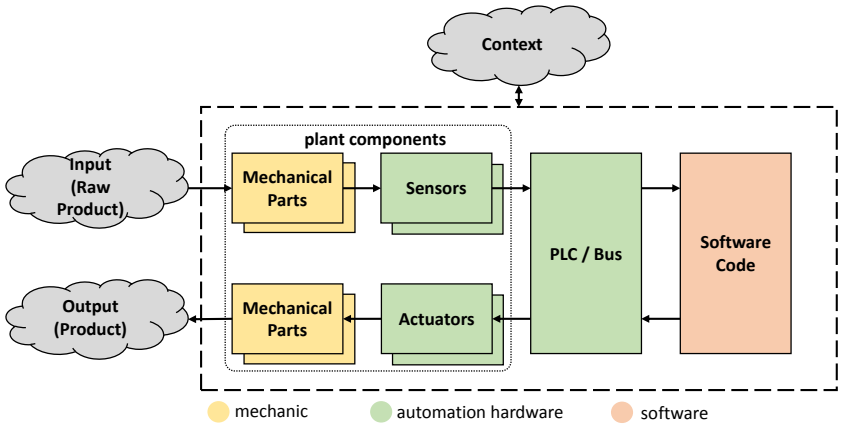

Figure 5. An abstract view on the different components of a cyber-physical production system

Improvement: A deviation with this trigger describes a cause based on a change that improves the production system. The objective and requirements of the production system remain the same, but the requirements are fulfilled better with the change. Within the production systems this could be an optimization of the structured hardware layout or an improvement of the precision of a product identification.

Requirement: These deviations go along modified requirements the production system must fulfil. For example, products must now be labelled by an additional machine or the characteristics of the product(s) are changed.

Triggered From:

Input: A cause arised by a product coming into the cyber-physical production system. This could be that the raw product is heavier than before or has another character.

Within: Any cause that is within the boundaries of the production system. This could be an individual mechanical, automation or software part or a plant component like a crane which consists of multiple of these individual parts.

Output: A deviation which is triggered from the needed (finished) product. For instance, the product requires a drill hole in the product which was not required before.

Context: A cause could also come be from the context of a production system. For example, the environment of the production system requires a specific change or, especially relevant in the world of cyber-physical systems the connection to another connected system or the human operator is changed.

Reaction Type:

Structure: This includes deviations which require a reaction in the hardware structure of the plant. For instance following an increase of the production volume, the

system must improve the capacity of a storage place in which the products are transferred.

System logic: The reaction affects the system logic, so that the system is afterwards operated differently. For example, when products need to be sorted, the system software logic must be changed because finished products are now transported to the existing ramps differently.

Production environment: Deviations of this type cause a reaction at elements of the environment that allow the production machine to be operated e.g. when the supplier or the automation hardware is modified or the logistic process following the production system is adapted.

Reaction At:[8]

Mechanical part: The reaction targets the mechanical part (yellow in figure 5) like a hardware ramp.

Automation hardware: Any deviation which requires a modification of the automation hardware as its reaction (green in figure 5) like a conflict in the bus system.

Software: The deviation requires a reaction that is done on the software code (orange in figure 5) , e.g. that the software execution is stuck.

Impact Effect:[9]

Timing properties: The impact of the deviation is observed on metrics for timing like the total production time or length of occupancy of the system.

Production volume: The deviation has an effect on the quantity which is, e.g., the production volume of a specific or all products.

Topology: The structure of the system is impaired by the deviation, e.g. if the connection or the routing between machines is affected.

Capability: Any deviation that has an effect that increases or reduces the capability (or functionality) of the production system. For example, a finished product has a different property or less product types can be stored in the system.

Impact On:

Input: A deviation can have impacts on the input product, e.g. less products can be processed in parallel.

Within (plant components): As it is mostly the case, any deviation of this category has an impact on the production system itself. Since the impact category focuses on non-functional properties the category is further expanded by considering each plant component separately. For instance, a crane occupancy is increased or a crane now can detect the weight of a product.

Output: Any impact which impairs the finished product as an output of the production system. For instance, a finished product is now labelled.

Context: The context is effected by an deviation. For example, when different monitoring values are provided to the connected systems or the human operator.

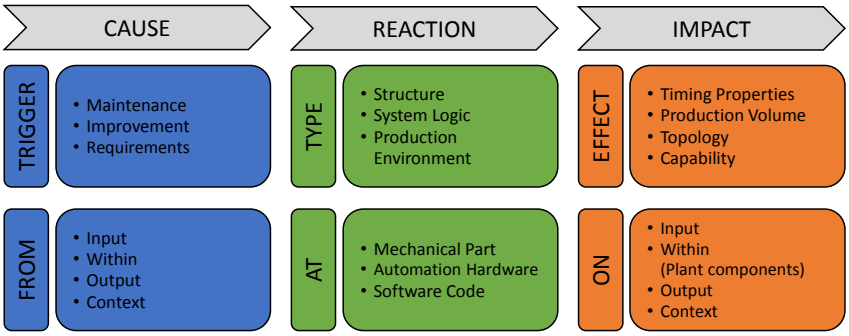

Figure 6. CRI-Model of a cyber-physical production system

The resulting CRI-Model for cyber-physical production systems is shown in figure 6. The CRI-Model is capable of describing malicious and evolutionary anomalies based on the generic production system model which includes the three involved disciplines, the in- and output as well as its context.

5. Applicability to Different Environments

In the following a mapping of practical data is done for the two considered domains with a focus on different objectives. The mapping shows that the CRI-Model is independent in terms of the focused domain and also in a high degree in terms of the targeted goal for with the CRI-Model is used.

5.1. Mapping of an Anomaly Dataset of Distributed Cloud Systems to the Taxonomy

As Cohen et al. state, the complexity of distributed cloud systems simply surpasses the ability of a human engineer to diagnose and correctly respond to every malicious behaviour [8]. To explore the applicability of the taxonomy to such complex systems, we decided to map anomaly reports from a large software company against the different categories of the CRI-Model. The reports consist of incident summaries and their impacts.

The objective of the mapping was to gain insight into which types of anomalies exist in the system, which would merit more thorough attention and which components are most often affected by a particular anomaly type. Under this objective the following three benefits of the anomaly categorisation arise:

1. Identifying common causes of anomalies and outages.
2. Finding areas where the monitoring of the system does not catch issues before a human does.

[8] The categories follows the commonly accepted classification of production system (as, for example, presented in [30]).

[9] The impact category focuses on non-functional properties of production systems as one of the main research points of the investigated evolution scenarios. The categories are built following an extension of the properties classification presented by Ladiges et al. [14].

3. A better understanding of the reaction chain that common anomalies experience.

Based on these identified fields, actions can be derived to improve quality as well as monitoring.

Data Foundations: The anomaly dataset consists of 51 reports that contain incident summaries with regard to root causes, problem descriptions, impact estimations, duration and type of detection (manual or by monitoring). The reports are Critical System Outage which implies that they only contain anomalies which had a very large impact on the clients of the cloud system, to merit reporting.

Mapping Example: Due to space limitations, we present two exemplary mappings:

1. The root cause was a bug which was introduced by a third-party vendor of a persistence component. This bug influenced how the queries were handled when a cluster of persistence nodes is not able to communicate with the primary node. A network issue was the trigger which is a hardware change within the system according to the CRI-Model, that forced all nodes to determine a new primary node. This procedure in turn triggered the bug and led to the CRI-Model reaction in the system that was connectivity issue on persistence components, since the individual nodes could no longer contact each other. The resulting impact was a delayed response time combined with an increase in error messages for the users (upstream) and required a temporary fix of upscaling the node cluster (costs for within).
2. The root cause for this issue was a feedback loop in the cloud system. Before the anomaly happened, the workers were under high load. In the CRI-Model the trigger can be sorted as usage pattern change, from upstream. The engineering team maintains a controlling system which stops and restarts workers that are operating under abnormal conditions. Before the incident the controlling system began rebooting worker instances which is a second trigger, in this case hardware change from the distributed system itself. Subsequently, other instances started getting a similarly high load (the reaction was therefore resource saturation) and were stopped and rebooted too (and connectivity issues), as were most instances that were brought up through the auto-scaling. The impact was an increased latency for the clients.

Applicability of Categories: To further verify the soundness of our categories, table 1 presents an overview on how many anomalies of the reports[10] could be mapped to which categories within our six dimensions. As previously explained any anomaly can consist of multiple triggers, reactions or impacts. However, in order to keep the mapping concise, for the validation regarding malicious anomalies we always only chose the most prominent category. For instance, when multiple triggers could be determined, we chose the category of the trigger with the strongest influence or the reaction category where most reactions of the report fit into. Of course for this kind of study only a small number of reports are available. Due to the low sample size for the usage analysis no significance level of the

[10] One report can contain more than one anomaly

CHANGE		REACTION		IMPACT	
Trigger	**65**	**Type**	**65**	**Effect**	**65**
Hardware	22	*Connectivity Issue*	24	*Data Integrity*	37
Software	21	*System Logic*	23	*Time Costs*	27
Usage Pattern	22	*Resource Saturation*	18	*Money Costs*	1
Triggered From	**65**	**Reaction At**	**65**	**Impact On**	**65**
Upstream	22	*Flow*	4	*Downstream*	0
Downstream	7	*Persistence*	18	*Upstream*	54
Within	36	*Worker*	43	*Within*	11

Table 1. Count of anomalies mapped to the categories of the CRI-Model for distributed cloud systems

Trigger x Reaction Type	***Software***	***Hardware***	***Usage Pattern***	***Sum***
Connectivity Issue	*1* / 2	*11* / 17	*3* / 3	22
System Logic	*7* / 7	*2* / 2	*2* / 3	12
Resource Saturation	*2* / 2	*0* / 1	*12* / 14	17
Sum	11	20	20	51

Table 2. Mapping of reported anomalies to the dimensions trigger and reaction. The first numbers indicate how many of the anomalies were detected by a human, while the other numbers indicate the total number of anomalies.

data can be achieved, but indications for validations can be derived which are given in the following.

According to the data in table 1 trigger and reaction seem to be both sensible. As expected for a distributed cloud system most changes are triggered by the system itself or the upstream. Reactions mostly happen on the workers which also held the most logic. As a consequence the CRI-Model shows that it is inescapable that metrics of workers fitting the most common anomalies are used for anomaly detection. Flows seem to be not affected that much or do not cause major breakdowns that are reflected in the reports. Regarding the impact, monetary costs and downstream are lacking mappings. However, the monetary impact of anomalies was usually not noted in the reports, and the impact on downstream services was also not communicated. In a different company this information might be available and very interesting to analyse.

Taxonomy Usage: In order to determine which types of anomalies might be particularly relevant to the system in question, and where anomaly detection should be implemented first, we picked two dimensions from the taxonomy (change trigger and reaction type), and sorted the reported anomalies into a cross-matrix (see table 2). We further distinguished which of these anomalies were detected manually (this means by an engineer or client) and which had been detected by monitoring. The first numbers indicate how many cases were detected by a human which ideally should never happen, if the human is a client or user.

Looking at table 2, it becomes clear that this system has most reactions concerning connectivity issues and resource saturation. Based on system operator knowledge, we know that most connectivity issues are triggered by sudden network problems, making it difficult to detect and mitigate. The resource saturation anomalies, on the other hand,

could be investigated further. This could indicate to the engineer, that anomaly detection should be implemented with a focus on hardware-level metrics, rather than error-logs, application-level metrics.

The analysis shows that beyond the vocabulary which the taxonomy provided, the CRI-Model can be used as a post-mortem analysis tool on an existing anomaly report dataset, and identify those types of anomalies most urgently needing attention. Based on this analysis, the CRI-Model can be employed to select and then cluster metrics and anomaly detection mechanism based on which anomalies are tackled.

5.2. Mapping of Evolution Scenarios of Cyber-Physical Production Systems to the Taxonomy

Due to its inherent interdisciplinary nature, detection of nonconforming behaviour is in cyber-physical production systems mostly model-based, which means that deviations are often identified by comparing the actual behaviour not just by metrics, but by specific behaviour models [13]. Here, domain-specific detection methods which observe the system behaviour on the software interface and comparing it with a model representation of the last known behaviour are used in order to support failure detection or evolution of the production systems [13]. To tackle the problem of deviations as well as deviation with anomalies, we divide evolution according to Vogel-Heusser et al. in long-term planned evolution and short time ad-hoc evolution [30]. Following both categories two different benefits of the CRI-Model become clear:

1. The CRI-Model provides a method to specify planned evolution a-priori as expected deviation to the previously correct specification. Classified deviations help to identify which adaptations in the system are needed and to better identify how and in which areas the system is affected.
2. When the adaption of the specification is omitted in ad-hoc changes, a classification of nonconforming behaviour with the CRI-Model serves as a documentation tool by describing the stage of the evolution process.

Data Foundation: The evolution scenario dataset is based on a Pick-and-Place-Unit that serves as a demonstrator for evolution of manufacturing systems. Detailed descriptions are available in [31] and in over 50 related publications (see e.g. [13, 15, 30]). The demonstrator includes 23 common evolution scenarios with 15 sequential and 8 parallel evolution steps which typically happen in production systems. The demonstrator as shown in figure 7 consists of five subsystems which are sequentially added by the evolution scenarios. The production system can handle three different products at its last evolution stage. The subsystems are:

1. A stack as a storage for products. The products are released in the system as its input.
2. A crane transporting the products from and to the other subsystems by picking them up with a vacuum rotary crane.
3. A stamp processing products by stamping them. The products are transported and released to the crane.
4. A conveyor belt which transports the products on a moving belt from the crane to output storage places.

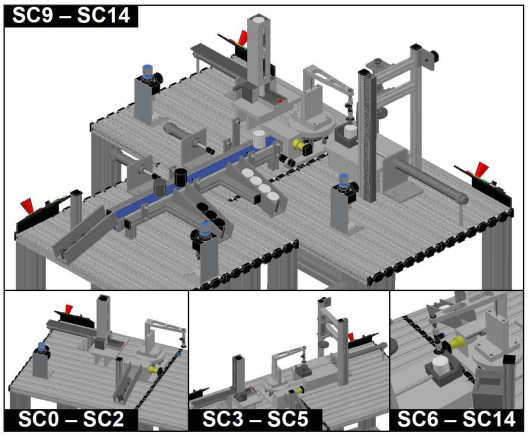

Figure 7. Case Study based on evolution scenarios (from [15])

5. Three output ramps as storage places which are filled by pushing products from the conveyor belt into the ramps.

To observe deviations between the specification and observed behaviour this article uses a self-documentation approach that rest on generating signal-based behaviour models (see [13]). The approach automatically learns and analyses the behaviour based on externally observable signal events. The learned models are specifically defined for automatic recognition of deviation (the machine state and material flow is learned) and their analysis regarding non-functional properties to evaluate evolution (see [14]).

Mapping Example: In the following two scenarios are exemplarily mapped to the CRI-Model:

1. Scenario 5 (according to [31]): The root cause of this evolution scenario is a planned change to improve the performance. The goal is to increase the workload of the production system because it is the bottleneck in the involved supply chain which means other systems like the disposal and logistic system require a higher production volume (requirement change from the context). To satisfy the requirement change, it was discovered that the behaviour of the crane is not optimal and therefore the software implementation needs to be changed which is a reaction in the system logic on the software code in the CRI-Model. The solution is an optimised crane behaviour which now checks during the stamping process if another product is available in the stack that does not need stamping. When this is the case, the crane transports this product to the conveyor belt first, instead of waiting for the stamping process to finish. The resulting deviation is according to the CRI-Model an increase in the performance (timing properties) and production volume on the crane component.
2. Scenario 7 (according to [31]): Due to market requirements the production system has the new demand to process a changed type of product. As a result the raw input, which are plastic tanks, are no longer only black, but also can be white which is a

CHANGE		REACTION		IMPACT	
Trigger	**23**	**Type**	**37**	**Effect**	**41**
Maintenance	2	*Structure*	13	*Timing Properties*	8
Improvement	8	*System Logic*	14	*Production Volume*	8
Requirements	13	*Production Environment*	10	*Topology*	10
				Capability	19
Triggered From	**23**	**Reaction At**	**52**	**Impact On**	**40**
Input	3	*Mechanical Part*	16	*Input*	3
Within	9	*Automation Hardware*	17	*Stack*	3
Output	2	*Software Code*	19	*Crane*	8
Context	9			*Stamp*	2
				Slide	3
				Conveyor Belt	11
				Output	5
				Context	5

Table 3. Count of scenarios mapped to the categories of the CRI-Model for cyber-physical production systems

requirement change of the input in the CRI-Model. Each input must be processed individually, but no sensor for detecting the colour is installed. The deviation is therefore a reaction in the production plant sensor topology, because neither the mechanic nor the automation hardware or software is capable of detecting the colour of the input. As a reaction an additional optical (infrared) digital sensor is installed at the stack to allow the system to detect the brightness of the input. Therefore, an increased capability of the stack and the input is the effect of the deviation in the CRI-Model.

Applicability of Categories: Table 1 presents an overview of how many scenarios could be mapped to which categories within our six dimensions. Whereas the change is always based on one trigger in this case study, due to the versatility of the non-conformance model the reaction and impact can have multiple types, effects and locations. It can be seen that most deviations are improvements or requirement changes and are mostly triggered by the system itself or its surrounding context. This is the result of the focused objective of the evolution scenarios, in which the normal maintenance activities as well as greater rewinds of the products are not directly targeted. Reaction and the involved different disciplines are quite well distributed along the scenarios. Here, the CRI-Model reveals that the scenarios are well chosen, because the balanced distribution is a good result regarding the demonstrator which explicitly tries to provide a wide range of different evolution steps. The impact on the non-functional properties has a peak for capabilities which results in the fact that most scenarios extend the production system with new elements that are mostly situated at the more complex plant components (crane and conveyor belt). Therefore, the CRI-Model shows that maintainability and changeability of the crane and conveyor belt should have the highest priority. These are deviations with anomalies which are detected by metrics of non-functional properties of the different plant components (see Ladiges et al. [13, 14] for the used approach).

Trigger x From / Effect	***Maintenance*** *n = 2*	***Improvement*** *n = 8*	***Requirement*** *n = 13*
Input	0 (0%)	0 (0%)	3 (23%)
Within	2 (100%)	6 (75%)	1 (8%)
Output	0 (0%)	0 (0%)	2 (15%)
Context	0 (0%)	2 (25%)	7 (54%)
Timing Properties	0 (0%)	5 (63%)	2 (15%)
Production Volume	1 (50%)	4 (50%)	2 (15%)
Topology	0 (0%)	4 (50%)	5 (38%)
Capability	2 (100%)	5 (63%)	11 (85%)

Table 4. Top: number (and percentage) of triggers within the given category of the column that are triggered from the given location of the row. **Bottom:** number (and percentage) of triggers that have an impact with the given effect.

Taxonomy Usage: To investigate the evolution scenarios and provide insights based on the CRI-Model we exemplary want to determine what kind of causes and reactions result in which impact on the system. Therefore, we first sort the deviations into a cross-matrix of the *trigger* with its *from* dimension and the *effect*. (see table 4). Improvements are mostly triggered by the system itself and equally affect non-functional properties which means that metrics of timing, production volume, topology as well as capability should be covered by the evolution detection method. In contrast, deviations triggered by changes that are caused by modified requirements are much more often initialized by a new need of the context of the production system and also sometimes by the product itself. Consequently, also most of the time (85%) only the capability of the system is enriched. Based on the CRI-Model deviations that are triggered by a requirement change have therefore less effects on metrics of the timing or production volume. Such an information can, for example, be used if the root change of a negative effect needs to be found.

Secondly, we analyse the deviations regarding the reactions, where the reactions happened at and what impact they had (see table 5). First, the interdisciplinary of the targeted domain of cyber-physical production system can be shown in the results, because most reactions happen currently at the mechanical parts, the automation hardware as well as the software. Only some reactions, for example, at the system logic impact just the software as one isolated category. The reactions which affect the structure and production environment often lead to a wider capability of the system while a reaction at the system logic can more often have an impact on other properties of the system. Therefore, when such a reaction is performed by an engineer, she should much more closely look if her modification does not lead to deviations on other components.

In general, mapping of evolution scenarios to the CRI-Model helps engineers to classify evolution, to analyse the evolution process and give indications which classes of evolution steps the production system undergoes during its life cycle. Compared to just using change requests, documenting the evolution along the CRI-Model provides an added value, due to an increased semantic meaning. Beside the reaction as the adaptation also the reason of the adaptation is documented in the trigger as well as the desired impact. These are valuable information, when, for example, a malicious timing on one component is detected. With the deviations ordered along the CRI-Model, evolution steps affecting

Reaction x At / Effect	***Structure*** *n = 13*	***System Logic*** *n = 14*	***Production Environment*** *n = 10*
Mechanical Part	11 (85%)	11 (79%)	7 (70%)
Automation Hardware	10 (77%)	9 (64%)	9 (90%)
Software Code	11 (85%)	14 (100%)	8 (80%)
Timing Properties	6 (46%)	7 (50%)	4 (40%)
Production Volume	6 (46%)	6 (43%)	4 (40%)
Topology	6 (54%)	7 (50%)	4 (40%)
Capability	12 (92%)	10 (71%)	10 (100%)

Table 5. Top: number (and percentage) of reaction with the characteristic of column on the given location in the row. **Bottom:** number (and percentage) of reactions that have an impact with the given effect.

the malicious timing of the component can be easily found as well as why this adaptation was done.

Applying the CRI-Model to cyber-physical production system for planned changes as well as ad-hoc changes has shown that the core aspects of the model can remain stable and the tailoring of categorization is simple and sufficient. The presented CRI-Model allows to classify deviations that are specified and planned in the same way as deviations resulting from ad-hoc changes on-side which are detected by an the detection approach of [13]. Further it allows to combine these deviations of specification and behaviour in a shared taxonomy with anomalies resulting as a malicious side-effect of changes or anomalies resulting from a rupture in the system.

6. Conclusion and Future Work

Based on research in anomaly detection and software evolution as well as publicly available outage reports of system failures and evolution scenarios, we introduce the CRI-Model, a taxonomy that maps non-conformances between behaviour and specification to six dimensions. Non-conformances are classified based on their cause, reaction and impact. This general mapping gives a good overview of different types of non-conformances for both anomalies resulting from raputres as well as deviations resulting from changes.

The model offers extensibility by specifying categories based on a model-based specification of component types. This allows users of the model to adjust it to their needs, while keeping a common understanding and language. The tailoring was shown and evaluated in two dimensions. First, the article distinguishes between different objectives by considering malicious anomalies and desired deviations as well as their overlap. Second, two domains are considered by tailoring the CRI-Model to distributed cloud systems and cyber-physical production systems. It was shown how the dimensions and domain specific categories provide a common taxonomy of these different but related areas of research. Further we outlined benefits of using the taxonomy like the selection of anatomy detection approaches and metrics or the documentation of an evolution process containing different kinds of scenarios in a shared taxonomy.

Further research could augment the model by adding a common set of metrics or detection techniques to each category of the taxonomy, easing the detection of specific

classes and studying which classes are notoriously hard to detect or have big impact to the company running the system. Further, a mapping of different metric types (system-level vs. application-level metrics for instance) and their contribution to effective detection with different types could create new insights and be a valuable tool. This could also include to formalise the abstract models to derive a formal representation of nonconforming behaviour that could be used to describe anomalies or changes in a system Finally, Tobergte et al. mention the need for an industry-wide repository of anomaly descriptions [28] and Ibidunmoye et al. request a publicly available datastore for performance datasets similar to the scenarios evaluated for evolution of production system [11]. Creating such databases with standardized formats (and dimensions, such as those from the CRI-Model) could foster a better understanding of anomaly types, confirm choices of anomaly detection approaches and allow a wider evaluation and comparison of those approaches.

To conclude, the taxonomy can be used for building and improving monitoring of anomalies, improving processes to remedy anomalies in case of system outages and is already been used to represent changes as evolution steps during runtime based on models shown in [13]. While a general purpose monitoring is desired, it may be hard to catch all kinds of nonconforming behaviour. Retroactively the CRI-Model helps in identifying areas that demand changes with the goal to improve quality of distributed systems. The CRI-Model lays the ground-work for a future mapping of different behaviour classes against existing detection and evolution approaches and creates a vocabulary and research tool for further work in the field of anomaly detection and software evolution.

Acknowledgments. This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future Managed Software Evolution.

References

1. Avizienis, A., Laprie, J.-C. & Randell, B.: Dependability and its Threats: A Taxonomy. in Proc. IFIP 18th World Computer Congress, 91–120 ((2004))
2. Babcock, C.: *Amazon's Vogels Challenges IT: Rethink App Dev (Interview)* (ed /www.informationweek.com) [Online; checked 22-Januar-2018]. Nov. 2012. `https://www.informationweek.com/d/d-id/1107599`
3. Baddar, S., Merlo, A. & Migliardi, M. Anomaly Detection in Computer Networks: A State-of-the-Art Review. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA) **5**.4, 29–64 (2014) (2014)
4. Buckley, J., Mens, T., Zenger, M., Rashid, A. & Kniesel, G. Towards a taxonomy of software change. Journal of Software: Evolution and Process **17**.5, 309–332 (2005) (2005)
5. Chandola, V., Banerjee, A. & Kumar, V. Anomaly detection: A Survey. ACM Computing Surveys **41**.3, 1–15 (2009) (2009)
6. Chapin, N., Hale, J. E., Khan, K. M., Ramil, J. F. & Tan, W.-G. Types of software evolution and software maintenance. Journal of Software: Evolution and Process **13**.1, 3–30 (2001) (2001)
7. Ciraci, S., van den Broek, P. & Akşit, M. A taxonomy for a constructive approach to software evolution. (2007) (2007)

8. Cohen, I., Chase, J. S., Goldszmidt, M., Kelly, T. & Symons, J.: Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. in OSDI. **4,** 16–16 ((2004))
9. Fluri, B. & Gall, H. C.: Classifying change types for qualifying change couplings. in Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on. IEEE, 35–45 ((2006))
10. Goltz, U. *et al.* Design for future: managed software evolution. Computer Science-Research and Development **30**.3-4, 321–331 (2015) (2015)
11. Ibidunmoye, O., Hernández-Rodriguez, F. & Elmroth, E. Performance Anomaly Detection and Bottleneck Identification. ACM Computing Surveys **48**.1, 1–35 (2015) (2015)
12. Isermann, R. Fault-diagnosis systems: an introduction from fault detection to fault tolerance (Springer Science & Business Media, (2006))
13. Ladiges, J., Haubeck, C., Fay, A. & Lamersdorf, W. Evolution management of production facilities by semi-automated requirement verification. at-Automatisierungstechnik **62**.11, 781–793 (2014) (2014)
14. Ladiges, J. *et al.:* Evolution of production facilities and its impact on non-functional requirements. in Industrial Informatics (INDIN), 2013 11th IEEE International Conference on. IEEE, 224–229 ((2013))
15. Legat, C., Folmer, J. & Vogel-Heuser, B.: Evolution in industrial plant automation: A case study. in Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE. IEEE, 4386–4391 ((2013))
16. Lehnert, S., Riebisch, M. *et al.:* A taxonomy of change types and its application in software evolution. in Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on. IEEE, 98–107 ((2012))
17. Maier, A., Niggemann, O. & Eickmeyer, J.: On the Learning of Timing Behavior for Anomaly Detection in Cyber-Physical Production Systems. in DX@ Safeprocess, 217–224 ((2015))
18. Mazel, J., Fontugne, R. & Fukuda, K. A taxonomy of anomalies in backbone network traffic. IWCMC 2014 - 10th Int. Wireless Communications and Mobile Computing Conf. 30–36 (2014) (2014)
19. Mirkovic, J. & Reiher, P. A taxonomy of DDoS attack and DDoS defense mechanisms. SIGCOMM Comp. Comm. Rev. **34**.2, 39–53 (2004) (2004)
20. Nielsen, J. Usability engineering (Elsevier, (1994))
21. Niggemann, O. & Lohweg, V. On the diagnosis of cyber-physical production systems: state-of-the-art and research agenda. Austin, Texas, USA, Association for the Advancement of Artificial Intelligence. (2015) (2015)
22. Parnas, D. L.: Software aging. in Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on. IEEE, 279–287 ((1994))
23. Pertet, S., Pertet, S., Narasimhan, P. & Narasimhan, P. Causes of failure in web applications. Parallel Data Laboratory December, 1–19 (2005) (2005)
24. Plonka, D. & Barford, P. Network anomaly confirmation, diagnosis and remediation. 47th Ann. Allerton Conf. on Communication, Control, and Computing, 128–135 (2009) (2009)

25. Reichert, K., Pokahr, A., Hohenberger, T., Haubeck, C. & Lamersdorf, W.: A Taxonomy of Anomalies in Distributed Cloud Systems: The CRI-Model. in Intelligent Distributed Computing XI (eds Ivanović, M. *et al.*) 247–261 (Springer, (2017))
26. Roth, M., Lesage, J.-J. & Litz, L. The concept of residuals for fault localization in discrete event systems. Control Engineering Practice **19**.9, 978–988 (2011) (2011)
27. Stammel, J., Durdik, Z., Krogmann, K., Weiss, R. & Koziolek, H. Software evolution for industrial automation systems: literature overview (KIT, Fakultät für Informatik, (2011))
28. Tobergte, D. & Curtis, S. Why Internet services fail and what can be done about these. Journal of Chemical Information and Modeling **53**.9, 1689–1699 (2013) (2013)
29. Vogel-Heuser, B. *et al.:* Selected challenges of software evolution for automated production systems. in Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on. IEEE, 314–321 ((2015))
30. Vogel-Heuser, B., Fay, A., Schaefer, I. & Tichy, M. Evolution of software in automated production systems: Challenges and research directions. Journal of Systems and Software **110,** 54–84 (2015) (2015)
31. Vogel-Heuser, B., Legat, C., Folmer, J. & Feldmann, S. Researching evolution in industrial plant automation: Scenarios and documentation of the pick and place unit. Technical University of Munich, Tech. Rep. TUM-AIS-TR-01–14-02. (2014) (2014)

Appendix A: Literature Sources used for the taxonomy

Barford, P., Kline, J., Plonka, D. & Ron, A. A signal analysis of network traffic anomalies. the second ACM SIGCOMM Workshop, 71–82 (2002) (2002)

Cheng, H., Tan, P., Potter, C. & Klooster, S. Detection and Characterization of Anomalies in Multivariate Time Series. Proc. SIAM, 413–424 (2009) (2009)

Cohen, I., Goldszmidt, M., Kelly, T., Symons, J. & Chase, J.: Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. in Proc. of the 7th symp. on Operating systems design and implementation. 4 ((2004))

Düllmann, T.: Performance Anomaly Detection in Microservice Architectures Under Continuous Change. Masterthesis (University of Stuttgart, 2017)

Dunning, T. & Friedman, E. Practical Machine Learning: A New Look At Anomaly Detection, 65 ((2014))

Fu, Q., Lou, J., Wang, Y. & Li, J. Execution anomaly detection in distributed systems through unstructured log analysis. Proce. - IEEE International Conference on Data Mining December, 149–158 (2009) (2009)

Goldstein, M. & Uchida, S. A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data. PloS one 11.4 (2016): e0152173 **11**.April, 1–31 (2016) (2016)

Gu, X. & Wang, H. Online anomaly prediction for robust cluster systems. Proc. - International Conference on Data Engineering, 1000–1011 (2009) (2009)

Guan, Q., Chiu, C., Zhang, Z. & Fu, S. Efficient and accurate anomaly identification using reduced metric space in utility clouds. Proc. - IEEE 7th Int. Conf. on Networking, Architecture and Storage, 207–216 (2012) (2012)

Gupta, M., Singh, A., Chen, H. & Jiang, G. Context-Aware Time Series Anomaly Detection for Complex Systems. Proc. of the SDM Workshop on Data Mining for Service and Maintenance, 14–22 (2013) (2013)

Hole, K. in Anti-fragile ICT Systems, 125–132 (Springer International Publishing, 2016)

Ibidunmoye, O., Hernández-Rodriguez, F. & Elmroth, E. Performance Anomaly Detection and Bottleneck Identification. ACM Computing Surveys **48**.1, 1–35 (2015) (2015)

Munawar, M., Jiang, M., Reidemeister, T. & Ward, P. Filtering system metrics for minimal correlation-based self-monitoring. IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems, 233–242 (2009) (2009)

Sharma, A., Chen, H., Ding, M., Yoshihira, K. & Jiang, G. Fault detection and localization in distributed systems using invariant relationships. IEEE/IFIP Int. Conf. on Dependable Systems and Networks **1,** 1–8 (2013) (2013)

Sheth, A., Doerr, C., Grunwald, D., Han, R. & Sicker, D. Mojo: A Distributed Physical Layer Anomaly Detection System for 802.11 WLANs. Proc. of the 4th int. conf. on Mobile systems, applications and services, 191 (2006) (2006)

Smith, D., Guan, Q. & Fu, S. An anomaly detection framework for autonomic management of compute cloud systems. Proc. - Int. Computer Software and Applications Conf. 376–381 (2010) (2010)

Takeishi, N. & Yairi, T. Anomaly detection from multivariate time-series with sparse representation. Proc. - IEEE Int. Conf. on Systems, Man and Cybernetics **2014-Janua**.January, 2651–2656 (2014) (2014)

Tan, Y., Nguyen, H., Shen, Z. & Gu, X.: PREPARE : Predictive Performance Anomaly Prevention for Virtualized Cloud Systems. in Distributed Computing Systems (ICDCS). Vcl, 285–294 ((2012))

Thottan, M. & Ji, C. Proactive anomaly detection using distributed intelligent agents. Network, IEEE October, 21–27 (1998) (1998)

Vogel-Heuser, B., Fay, A., Schaefer, I. & Tichy, M. Evolution of software in automated production systems: Challenges and research directions. Journal of Systems and Software **110,** 54–84 (2015) (2015)

Vogel-Heuser, B., Legat, C., Folmer, J. & Feldmann, S. Researching evolution in industrial plant automation: Scenarios and documentation of the pick and place unit. Technical University of Munich, Tech. Rep. TUM-AIS-TR-01–14-02. (2014) (2014)

Wang, T., Zhang, W., Wei, J. & Zhong, H. Fault Detection for Cloud Computing Systems with Correlation Analysis, 652–658 (2015) (2015)

Appendix B: Reports of System Outages

Company	*Date*	*Source*
HealthCare.gov	01.10.2013	https://www.theguardian.com/world/2013/oct/30/ health-secretary-sebelius-healthcare-website-hearing-live
Motorola	02.12.2013	https://www.cnet.com/news/motorola-delays-moto-x-cyber-monday-deal-after-site-crash
Bank of America	01.02.2013	http://www.cbsnews.com/news/bank-of-america-web-portal-back-online-after-outage
Yahoo	09.12.2013	http://www.theverge.com/2013/12/11/5201146/yahoo-apologizes-for-email-outage
AWS	13.09.2013	http://www.usatoday.com/story/tech/2013/09/13/amazon-cloud-outage/2810257/
Google Amazon Microsoft	19.08.2013	http://blog.smartbear.com/performance-testing/ any-given-monday-google-microsoft-and-amazon-all-experience-outages/
Nasdaq	22.08.2013	http://www.networkcomputing.com/government/ nasdaq-outage-explored-7-facts/276050510
LinkedIn	23.10.2013	http://www.onlinesocialmedia.net/20131023/ linkedin-outage-today-follows-facebook-going-down/
LinkedIn	18.06.2013	https://techcrunch.com/2013/06/19/linkedin-outage-due-to-possible-dns-hijacking/
Verizon	18.06.2013	http://www.datacenterdynamics.com/content-tracks/security-risk/95538.fullarticle
Twitter	19.01.2016	http://www.techradar.com/news/world-of-tech/ twitter-is-down-for-most-people-right-now-1313338
Microsoft	18.01.2016	https://www.theregister.co.uk/2016/01/25/office
Salesforce	03.03.2016	http://www.v3.co.uk/v3-uk/news/2449449/ salesforce-suffers-cloud-outage-and-service-disruption-in-europe
Salesforce	09.05.2016	http://www.zdnet.com/article/ circuit-breaker-failure-was-initial-cause-of-salesforce-service-outage/

Table 6. The List of system outages reports from (software) companies

Christopher Haubeck is a researcher in the Distributed Systems research unit in the department for informatics of the Hamburg University. His main scientific interests are software for distributed and cyber-physical systems, architectures for knowledge carrying software, coevolution of runtime artefacts and simulation.

Dr. Alexander Pokahr is head of the Industrial Data Processing and Systems Analysis chair at the Helmut-Schmidt-University / University of the Bundeswehr Hamburg. Current research interests include development approaches for large-scale, intelligent cyber-physical systems as well as agent-oriented control of autonomous mobile robots. In addition, he is acting as CEO of Actoron GmbH, which he co-founded.

Kim Reichert graduated at the University of Edinburgh with a master?s in design informatics and wrote her second Master Thesis on the topic of anomaly detection in distributed systems at the University of Hamburg. She is working at the Adobe Systems Engineering GmbH at the Adobe Cloud Platform. Beyond her daily work, her focus lies on monitoring and automated anomaly detection of the platform.

Till Hohenberger is working on the Adobe Cloud Platform - Content Services in Hamburg, which is the backend for most Adobe cloud-based products at the Adobe Systems Engineering GmbH. He develops and maintains multi-regionally distributed services to manage access control to cloud resources.

Winfried Lamersdorf is a professor in the Informatics Department of Hamburg University and head of the Distributed Systems research unit. His main scientific interests are in the areas of system software for distributed systems, service-orientation, middleware, agent- and component-oriented, autonomous, self-organizing and mobile systems as well as related applications from e-business / e-services via business process management up to logistics and production automation.

Received: January 26, 2018; Accepted: September 4, 2018.