Fleet-based System Health Assessment: Reasoning about Change and Differences

Michael Borth[1]

[1]*ESI (TNO), High Tech Campus 25, 5656 AE Eindhoven, The Netherlands*
Michael.Borth@tno.nl

ABSTRACT

Monitoring and managing the health of technical systems with advanced diagnosis and prognosis benefits from fleet analytics: insights on the degradation of other but similar systems help, e.g., to forecast actual issues for predictive maintenance as does detecting and correcting anomalies in usage profiles helps to prevent undue wear and tear. Successes in this field usually depend on the similarity of the fleet's systems: although not necessarily equal, they need to have alike key characteristics, e.g., in the way they age, such that observations on one or many systems constitute an expectation for others.

We introduce fleet-based system health assessments that complement such approaches by reasoning on differences, e.g., those introduced by interventions like upgrades. Given that such change is the only constant for many of today's complex systems, we believe that our addition to health assessment is necessary to cope with the variation in systems that fleet operators face over the lifecycles of all systems.

Technically, our approach is based on probabilistic reasoning. The related modeling and inference techniques allow us to incorporate insights from past observations in individual systems, but also from comparable systems and, at the same time, they permit comparisons between parts of the fleet and estimate possible effects of existing differences. Our work shows that especially such comparisons are valuable for fleet health management over long lifecycles that is typically linked to continuous improvement processes.

1. SYSTEM HEALTH MANAGEMENT FOR FLEETS

System Health Management (SHM) assesses the current or future health state of a system to increase its reliability, availability, and dependability. Based on diagnosis and prognosis techniques that analyze available sensor data, SHM typically aims to shorten the time-to-repair in the event of failures and to prevent such failures and the related down time altogether with predictive and preventive maintenance. Technically, recent advances in the SHM are mainly based on the fusion of Internet-of-Things (IoT), i.e., the extension of networked connectivity into physical devices and sensors, with advances in data science that allowed faster and more accurate predictions.

Due to this progress, predictive maintenance became a major business trend, as identified, e.g, by Gartner (Berthelsen, 2018), even though Mulders et al. (2018) analysis that many companies remain in early stages of implementing it yet holds in 2020 according to our experience. Still, given a framework of available resources and operational demand, it became possible to better plan the most efficient use of systems, altogether lowering the total cost of ownership (TCO). Such a framework may exist on several levels: (i) for a single system, provided that its tasks vary in duration or intensity such that fitting scheduling and maintenance plans based on health information increase its utilization; (ii) for a system of systems (SoS), e.g., a factory, which relies on individual systems, but may also include alternate systems that can complete a task and whose performance (throughput, etc.) depend on their health state, thus offering another layer of optimization; and (iii) for fleets of systems.

For the latter, we know that automotive, shipping, and avionic industries benefits from monitoring and managing the health of assets with advanced diagnosis and prognosis that draws from fleet analytics. In these industries, both asset owners and manufacturers have a vested interest to work cooperatively and thus the organizational and technical means to pursue SHM approaches where insights on similar systems help to increase the precision of the reasoning about a specific system of interest.

In automotive, e.g., this led, among many other cases, to the success of services like Fleetboard (www.fleetboard.info) or early publications like the work of Wirth and Reinartz (1996) of Daimler and the cross-industry standard for data mining (CRISP-DM).

Methodologically, such work often depends on the degree of similarity of a fleet's systems: although not necessarily equal, they need to have alike key characteristics, e.g., in the way they age, such that observations on one or many systems constitute an expectation for others. This requisite, however, is often violated, as we elaborate in Section 2 as the first contribution of this paper. This insight then motivates our approach to complement system health assessments by fleet-based reasoning on similarity *and* differences, which sets the main contribution of our paper.

We illustrate this main part of our work in two sections. In Section 3, we initially refer to earlier work where we investigated the impact of different states of systems that arise from upgrades and updates. We then show that even basic probabilistic reasoning can determine root causes for performance loss once sequences of interventions or maintenance actions are analyzed on fleet level. Exploiting more subtle differences between systems, however, requires elaborate reasoning that matches current observations with past data both within the timeline of a system and across systems while estimating effects of existing differences. We develop this in Section 4.

We close with an experimental evaluation of our approach in Section 5 followed the conclusions in Section 6.

2. SIMILAR – BUT SIMILAR ENOUGH?

System differ among each other, e.g., regarding their configuration or feature set, usage patterns, or even production quality and thus wear and tear, but they also change over time, e.g., due to maintenance actions, software updates, hardware upgrades, or switches in their behavior.

Many techniques for predictive or preventive system health management assume that these differences do not matter for the prediction if it is possible to find a *condition indicator value* (CIV) that can be measured or computed to define a signature curve for the investigated wear and tear. Such a *remaining useful life* (RUL) estimate relationship to the condition indicator is shown in Figure 1 and explained in, e.g., (Baru, 2018) or the review article of Xiao-Sheng Si et al. (2011). Often enough (but not always, see below), this approach works, especially for mechanical wear and tear, where vibration signals typically indicate impeding breakdowns with little or no dependence to other factors.

In those cases, the RUL estimate can be done based on similarity between systems, such that an incoming time-series of the CIV of a system is compared to those of other systems and the most similar known series is used to predict the systems future wear and tear behavior.

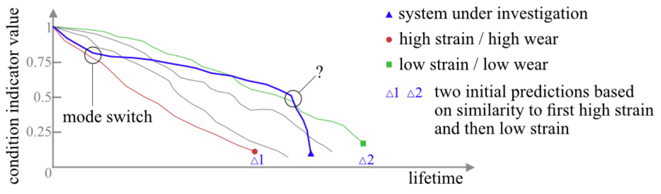


Figure 1. Switching behavior in RUL estimates. (from Borth and Barbini, 2019)

In Figure 1, however, we investigate a system that at first runs on course for prediction 1, but changes, showing later a similarity leading to prediction 2, before changing again and ending with an abnormal behavior. This investigation relates to an adaptive system's ability to perform under different load scenarios and in different environments – imagine a seafaring vessel that leaves a high strain environment to prevent critical mission failure (mode switch), but which initial wear leads to an accelerated tear that fails to manifest early in the CIV. The latter happens as the low strain phase induces other characteristics in the CIV, especially as the adaptive system enacts compensating behavior that hides indicators normally visible within component interactions.

The importance of component interactions for wear and tear was one of several factors that led us to seek novel ways for system health management in our work on complex high-tech system. We also needed to ensure a system's ability to perform, which is a stronger requirement than a prevention of breakdowns, as we explained in (Borth & Barbini, 2019).

Furthermore, we saw that differences in features can matter for predictive health management in unforeseeable and nonconstant ways. For our vessel investigation, e.g., the second change point looks very different in systems with auxiliary subsystems that would prevent the hidden tear – but without the initial phase of high wear, this feature would not impact the RUL. Additionally, we face the complication that all systems we investigate undergo irregular updates in their embedded software – changing system behavior and thus, potentially, wear and tear – and even upgrades of parts.

In total, this leads to the conclusion that any similarity we seek to exploit during fleet analytics is temporal at best; phrased differently, we state that past similarity may be no indication of future similarity and even seemingly similar systems might not be similar enough to transfer insights from one system to another if the context is disregarded.

This has grave consequences for any purely data-driven approach: as we illustrated in (Borth & van Gerwen, 2018), even lots of data is not enough for a similarity-based analysis if the data set is broken down into too many compartments as the number of cases within these becomes too small. Consequently, we seek to exploit not only similarities, but also knowledge about differences between systems in our fleet analytics, as these opposing aspects can complement each other, as we show below.

3. PROBABILISTIC HEALTH ASSESSMENT

The fleet analytics we introduce in this article are set within a line of work in which we use probabilistic reasoning for the assessment of a system's health and readiness to perform an upcoming mission.

To briefly recap what we and others stated before, system health assessment is, like any diagnosis problem, the task to deduce the likelihood of an internal state that is not directly observable from observations that are possible, like sensor data. These observations are typically of limited accuracy or even temporarily missing. The reasoning itself is then made under several assumptions, e.g., the a priori likelihoods of component failures and the impact of the environment as well as internal and external interactions. All in all, this summarizes as reasoning under uncertainty and probabilistic reasoning, e.g., with the Bayesian networks we deploy, is a leading methodology for it. Pearl introduced Bayesian networks in (1986) and they showed their performance early on in a range of applications (Heckerman, 1995).

We consider them to be especially suited for building diagnostic models of systems as they are one of the few techniques that combine knowledge- and data-driven modeling, as e.g., Jensen illustrated (2007). In recent years, the generation of suitable Bayesian networks saw great improvement as the progress in data science enhanced the latter, while probabilistic programming techniques eased the former (Hardesty, 2015).

For us, this led to the systematic approach detailed in (Borth & Barbini, 2019), but we also refer to (Ricks & Mengshoel, 2009). In summary, we first generate a knowledge-based directed graph in which nodes represent system variables, e.g., health states of components or sensor readings, and edges encode dependencies between variables. The parameters of these dependencies, i.e., the conditional probabilities which altogether encode a joint probability distribution over all variables, are set in a data-driven step for which many techniques exist, see e.g., (Barber, 2012).

To diagnose a system using a Bayesian network, we enter observations as evidence, i.e., we instantiate the respective nodes to the observed state. Bayesian reasoning then updates all other variables given the conditional probability distributions that specify the relationship between variables. This provides the marginal probabilities for all possible states and thus the likelihoods of component failures and system malfunctions or losses of performance.

The latter aspect, i.e., the ability to reason about a system's performance is crucial for the fleet analytics we introduce here, as Bayesian reasoning under uncertainty is inherently bi-directional: it allows us to infer expectations about a system's behavior given a cause, e.g., the impact of a specific feature, but also to infer possible explanations or root causes for an observed behavior, including a loss of performance.

3.1. Reasoning on Fleet Observations

One of our early results in exploiting observations from a fleet of systems to determine the root cause of performance loss in operational systems was the so-called ESI System Data Demonstrator (www.esi.nl/system-data-demo/). Figure 2 shows the relevant step here, where a Bayesian network analyses a fleet of 65 machines, which partially received hardware and/or software upgrades or updates. The red color and small cross icons indicate that many systems have performance issues, but the distribution is not clear-cut: there are, e.g., many systems with, but also many systems without issues in the left and in the right fields, which list systems without and with software upgrades respectively.

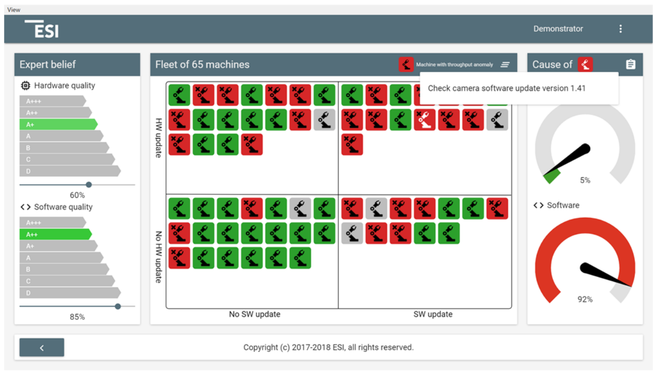
However, our probabilistic health assessment brings the ratios of systems with or without issues in all four quadrants into relation with the prior expert belief on the quality of the hardware and software development (left) and arrives at the conclusion that a recent software update is to blame (right). The structure of the necessary Bayes net consists of only two nodes for root causes, i.e., an ill effect by either software or hardware changes and nodes for the affected parts of the fleet in which we enter the observed ratios of systems with performance issues.

While this fleet-based analysis of the impact of hardware or software changes works well, we see two major limitations:

- The performance of systems and thus the major piece of evidence for the reasoning is judged by a comparison against a global baseline.
- The reasoning is static.

We can, e.g., not easily integrate that a software update only had a marginal negative effect on a system if that system already performed badly and the change pushed it beyond the set threshold, nor can we exploit a sequence of changes, e.g., if a system receives a software update without issues but then takes a performance hit during a hardware upgrade.

Taking these points as motivation, we introduce more elaborate health assessment considerations below.



Check camera software update version 1.41

Copyright (c) 2017-2018 ESI, all rights reserved.

Figure 2. ESI System Data Demonstrator.
Bayesian reasoning diagnoses a software patch hit.

3.2. Causal Reasoning on Circumstances

First, we investigate how circumstances like load or the environment factors impact system states. We specify such factors according to expert knowledge and introduce the dependencies between the factors and system behaviors including component wear and tear fitting to known facts or observed data. As the resulting Bayes net encodes the joint probability distribution of all its variables, we are then able to compute any marginal distribution of interest, e.g., the health state of a component, given any set of circumstances or other factors. In this, we can factor in the strain and thus wear and tear of scenarios or – and that is equally important – factor it out.

This modeling exploits that we can reason along causal lines within Bayesian networks. Doing so is an asset for fleet analytics, as it allows for a transfer of insights: if a causal effect is established and specified in a part of the fleet, it will hold in all systems where it can occur – even if it was not observed in those systems yet. To understand the foundation of this transfer, we refer to (Pearl, 2009) for the theory on how causality introduces modularity and composability into probabilistic modeling.

In our work, we use causality to reason on system aspects like their age, usage, configuration, or features:

- knowing the Weibull distribution that describes the failure likelihood of a component *given* its lifetime allows to assess the impact of age *together* with wear and tear of load caused by circumstances;
- features and configurations *together* with their usage profiles determine system load and thus wear and tear and affect health assessments according to Failure Mode and Effect Analyses and load distribution models.

3.3. Similarity Re-visited

Using causality and thus expert knowledge to transfer insights between parts of the fleet partially counters the compartmentalization we described at the end of Section 2.

Furthermore, it addresses the first of the two concerns listed above, i.e., the limitations that stem from comparing health indicators only against a global baseline, as it allows us to tune a comparison to the given circumstances: We can, e.g., conclude that a system which seems to struggle is actually doing well given that it operates in harsh conditions.

This allows us to re-visit the question of similarity central to many RUL techniques: instead of using a fixed computation for the condition value indicator, we factor in the differences between the systems of the fleet, effectively replacing the curves shown in Figure 1 by those we compute under a set of common assumptions.

This results in an adapted prediction of RUL from the lines Δ1, Δ2 in Figure 1 towards the Δ1', Δ2' shown in Figure 3.

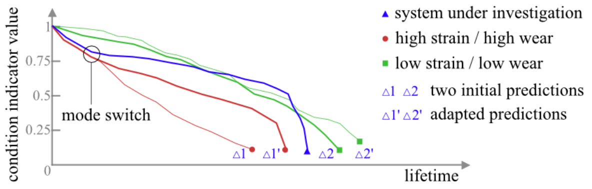


Figure 3. RUL estimates adapted to switching behavior.

The estimate has less uncertainty as the new predictions are closer to the reality of the system under investigation. Computationally, it stems from so-called counterfactuals that pose a 'what-if' question that allows us to ask how the curves would look like following Bayesian inference within the algorithmic pattern first laid out in (Balke, 1994):

1. Update the probability of unobserved factors given the observed evidence according to the Bayes rule.
2. Perform the action in the model that changes the model towards the target configuration, i.e., the counterfactual.
3. Predict RUL in the modified model.

The effort to compute counterfactuals along these three steps is significant, but it is reduced greatly if there is a single causal model that can accommodate all factors, i.e., there is no factor which is true for parts of the fleet but cannot be true for others.

In this case, which often holds for our work, there is no contradiction between the factual world and possible actions of interest in the interventional level, e.g., mode switches or software updates. Consequently, we do not need to modify the model during the RUL computations and can, instead, insert assumptions as observations, falling back to simpler Bayesian inference for which ample tooling exists.

3.4. Dynamic Reasoning over Time

The final building block we introduce for our health assessment approach is to reason about change over time.

With Bayesian networks, reasoning over time is typically implemented in the form of Dynamic Bayesian Networks (DBN), as introduced by Lerner et al. (2000) for fault detection and diagnosis in dynamic systems, where the authors used the concept of time-slices to model and then reason over time.

This and subsequent works transform the diagnosis problem into the task of tracking the system state and then to draw conclusion from the way observed behavior deviates from the expectations. The latter is supported by using different DBN for normal behavior and error behaviors, as shown, e.g., in (Roychoudhury et al. 2008), such that the network which fits the observed behavior the closest provides the diagnosis while the other DBN are disregarded.

We extend these ideas towards fleet analytics and reasoning about change by introducing a time-stepwise construction of a fleet analytics network that tracks the changes that happen to systems within the fleet. Extending the example in Section 3.1 for the ESI System Data Demonstrator, we show this in Figure 4, for a software update *sequence*, i.e., a stepwise roll-out of a new software version that is, over time, used by more and more systems of the fleet.

In the reasoning of this growing network, we use observations regarding the performance of individual systems to estimate the quality of a software patch, wherein a performance degradation supports the hypothesis of a software issue while the opposite serves as counter example. As we extend our reasoning over time towards any system that uses the updated software version, we always use all available data while we run, i.e., past and present, and continuously update our beliefs over all systems and their past and actual state. This extension over time is the difference to the reasoning used in Section 3.1 – it forms a dedicated analysis that delivers immediate insights and becomes more certain over time as evidence for a software patch hit grows or decreases.

3.5. Building Probabilistic Health Assessments

This far, we introduced a set of building blocks for system health analytics: diagnostic assessments that follow causal modeling principles to ensure modularity and composa-bility; transfer of insights between systems with adaptive similarity computations that match a system's behavior to the behavior of other systems that we either observed or that we would expect to observe given similar circumstances; and dynamic reasoning over time that mirrors the changes that happen to the systems. We realize all these building blocks with probabilistic reasoning, specifically Bayes nets, within one modeling approach. This ensures a consistent assessment without conceptual breaks.

Implementation-wise, we use the probabilistic programming language Figaro (Pfeffer, 2016) for this, which allows us to mix object-oriented and functional programming to define Bayes nets which are then piece-wise constructed to cover the dynamics and changes to systems and the fleet.

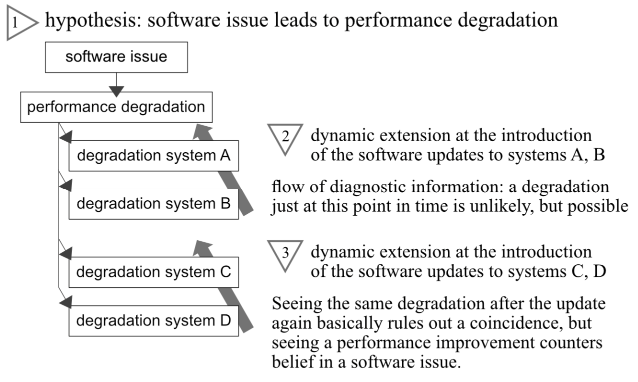

1
hypothesis: software issue leads to performance degradation
software issue
performance degradation
degradation system A
degradation system B
degradation system C
degradation system D
2
dynamic extension at the introduction of the software updates to systems A, B
flow of diagnostic information: a degradation just at this point in time is unlikely, but possible
3
dynamic extension at the introduction of the software updates to systems C, D
Seeing the same degradation after the update again basically rules out a coincidence, but seeing a performance improvement counters belief in a software issue.

Figure 4. Reasoning over time and multiple systems I.

4. FLEET-BASED SYSTEM HEALTH ANALYTICS

Using the building blocks introduced above, we now realize fleet-based system health analytics that serve three purposes:

- to determine the health of an individual system based on expectations and insights that originate from the system's own past behavior, other systems of the fleet, and from the specifications of the system's functions;
- to detect faults or performance loss introduced by changes like software issues due to update processes;
- to detect execution faults, i.e., certain transient errors caused by failures in the orchestration of systems and functions, e.g., if a need to correct a process step is not communicated in time.

One of our application domains here is process control where a complex software function is used to keep manufacturing equipment within its operational window. Software like this is typically used to counter physical effects that make the equipment drift out of their optimal work conditions, thus controlling an aspect of the process in order to keep systems operational without expensive physical calibration.

With this link into process control, our health analytics sets out to outperform standard process control techniques, like those based on Western Electric Rules (WER) introduced by Western Electric in (1956), but also to offer more insights than the simple fact that something is beyond 'the normal'.

Our success criteria are thus timely detection of all errors and faults, optimally on the time-step in which they show in the data, thus no false negatives, and few to zero false positives.[1] The motivation for these goals is that failing to run processes correctly incurs heavy costs, while false positives lessen the trust in the health analytics.

We show that we meet our goals in Section 5, where we describe experiments for a small fleet of six systems. Larger fleets pose no undue challenge, as we explain at the end of Section 4.

4.1. Probabilistic Health Monitors

The first components to realize for our fleet-based system health analysis is a set of probabilistic health monitors that each run a concurrent analysis of an individual system. The task of these monitors is to check a system's behavior, visible in its response to an input, against expectations.

Providing us with a probabilistic measure of the match between observations and expectations, the monitors show similarity to Western Electric Rules in the sense that they state if something is unexpected or unlikely.

[1] We consider it noteworthy that statistical techniques like the WER cannot deliver such a feat as they have an inherent delay and must accept individual data points that are 'off' as outliers which are bound to happen. Their advantage is thus not in their absolute performance, but in their performance given the simplicity of their use.

Their composition, however, is more sophisticated, as they compute the following three comparisons:

1. The response of a system to an input is compared to the system's own earlier responses.
2. The response of a system to an input is compared to earlier responses of other systems in similar circumstances; i.e., we use the causal reasoning to adapt the other systems' actual responses towards the situation at hand to ensure sufficient similarity.
3. The response of a system to an input is compared to its designed function, i.e., to the response we expect under perfect conditions (no noise or interference, etc.).

These three steps are illustrated in Figure 5. As shown, we run the first two comparisons (1, 2) in a similar way:

Given that there are typically no past values for an identical input x, we consider a number of (output y_i | input x_i) value pairs within the data sets of past observations that are relevant for checking the current observation (y | x).

Taking these value pairs, we produce a weighted combination y' of the output values $\{y_1,\ldots,y_n\}$ to formulate the output we expected from the past. The weights used for this are based on the distance between x and x_i in the same way that we then use to compute a measure for the match between the expected value and the observed one, where we use the probability defined via the normal distribution around y'

$$p = \text{cumulative probability outside } y' \pm |y' - y|$$

$$\text{e.g.,} \quad p = 2\times 2.3\% = 0.046 \text{ for } y = y' - 2\sigma$$

$$\text{e.g.,} \quad p = 2\times 50\% = 1 \text{ for } y = y'$$

Within the second comparison of a system to other systems of the fleet, we average over all match values (2 B).

The comparison of an observed value y with the designed function $y^* = f(x)$ uses the same match evaluation as the previous steps (3). However, we find it good practice to also consider rules defined by experts to adjust our result in cases where the response of the system is out of bounds, i.e., clearly beyond the operational window, resulting in a penalty that discriminates against identifiable faults.

In total, this might lead to a flow of information during the fusion of the three outcomes that resembles an argument, e.g., stating that we have seen numbers like those observed before, but with the final check against the function informing us that these are still wrong.

Seeing that we handle process volatilities and variety between systems together with the functional requirements, we value such 'disagreements', as they usually point towards interesting happenstances for further investigation.

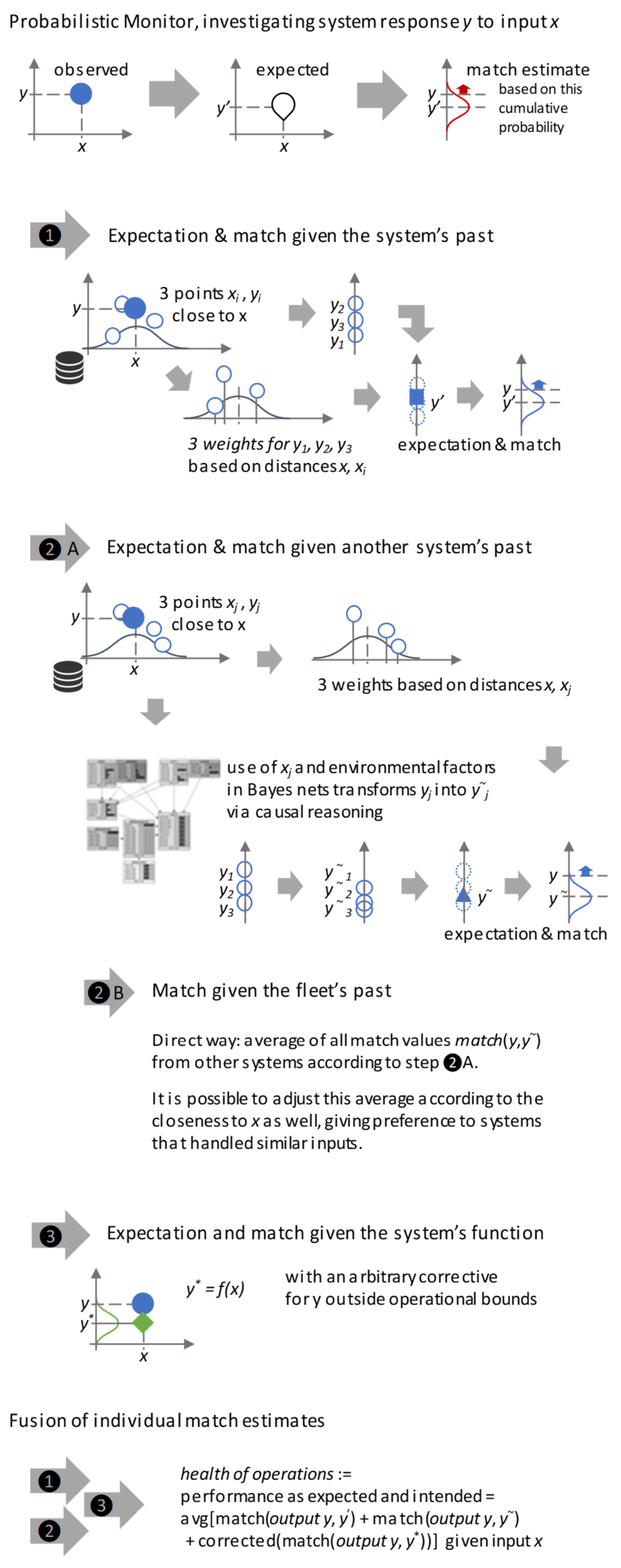

Probabilistic Monitor, investigating system response y to input x
observed
expected
match estimate
based on this cumulative probability
y
y′
x
1 Expectation & match given the system's past
3 points x_i, y_i close to x
y_2 y_3 y_1
y′
3 weights for y_1, y_2, y_3 based on distances x, x_i
expectation & match
2 A Expectation & match given another system's past
3 points x_j, y_j close to x
3 weights based on distances x, x_j
use of x_j and environmental factors in Bayes nets transforms y_j into $\tilde{y}_j$ via causal reasoning
$\tilde{y}_1$ $\tilde{y}_2$ $\tilde{y}_3$
$\tilde{y}$
expectation & match
2 B Match given the fleet's past
Direct way: average of all match values match(y,ỹ) from other systems according to step 2A.
It is possible to adjust this average according to the closeness to x as well, giving preference to systems that handled similar inputs.
3 Expectation and match given the system's function
$y^* = f(x)$
with an arbitrary corrective for y outside operational bounds
y^*
Fusion of individual match estimates
health of operations := performance as expected and intended = avg[match(output y, y′) + match(output y, ỹ) + corrected(match(output y, y^*))] given input x

Figure 5. Flow for probabilistic health monitors.

4.2. Investigations of Interventions

Next to assessing state and performance of individual systems, we use the probabilistic health monitors to investigate interventions, i.e., man-made changes to systems or systems-of-systems, within the fleet.

This application, which we found particularly useful to check software updates for non-intended side effects, follows the ‘Reasoning over time and multiple systems’ approach that we illustrated in Figure 4, but now uses the probabilistic health monitors to judge the degradation of a system’s performance, replacing the ‘degradation system X’ nodes. Consequently, we feed the software quality analysis that runs across the fleet and over time insights about individual machines which are, in turn, based on past data of the same machine and of other machines, but also on the function of the software. We depict this in Figure 6.

We found this approach particularly valuable for domains with small signal-to-noise ratios. We saw this, e.g., in the process control scenario, where drift corrections computed by the software are small in comparison to the noise introduced by process variations and measurement errors: the software update is supposed to improve the accuracy of corrections, but as its introduction upsets the production flows, we need to account for a multitude of temporal effects that also differs per machine, but the combination of the different aspects of the probabilistic health monitors makes the analysis robust against volatile circumstances.

In practice, we saw this stabilization of the analysis also improved by the explicit notion of supporting or weakening a hypothesis about a software update’s performance within probabilistic reasoning that encompasses the whole fleet. As illustrated through the graph in Figure 6, the software update is either good or bad, but it cannot be the one for one system and the other for another, even though local observations might provide that impression for a while.

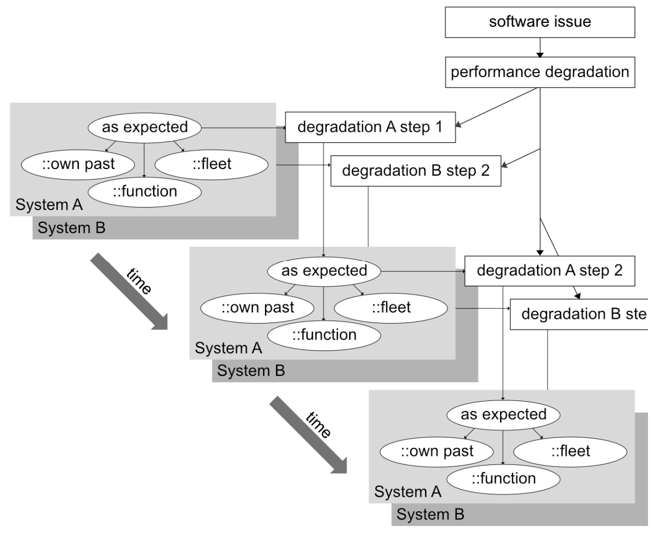


Figure 6. Reasoning over time and multiple systems II.

4.3. Detection of Execution Faults

The detection of execution faults is the last application we realize with the probabilistic health monitors. These faults originate from failures to orchestrate systems in a system-of-systems setting or, e.g., from missed race-conditions within a complex system. They are often individual in their nature and thus not per se part of fleet analytics.

Our experience, however, shows that it is helpful to transfer knowledge between systems of a fleet when building detectors dedicated to execution faults: if a certain error type leads to a specific behavioral pattern in one system, we extract that pattern and use it to detect (and subsequently correct) that error in other system. We rely again on causal reasoning to factor out differences between individual systems and search, e.g., for a missed execution of a drift correction in very differently scaled trend analyses if the systems differ in the volatility of their process stability.

We realize the detection of an execution fault pattern with a sequence of the probabilistic health monitors: the first monitor starts with the current observation to provide a match estimation for an expected system response given the assumption of an error, followed by subsequent monitors that all compare the actual response in that time step to that which would follow for that time step if the error happened. This tracking approach gives us a sequence of assessments that either confirm or reject the hypothesis of the execution error over time.

In practice, we build such elaborate detection mechanisms for execution faults only in black or grey box situations, i.e., if we do not have full observability of input and output of a system component with full knowledge of its inner functions. The fusion of expectations from expectations and from past observations within the fleet allows us to track behavior even with such limited knowledge, while a sequence of functional monitors is sufficient for white box scenarios.

4.4. Optimal Fleet-size for Health Assessments

In our work, we saw benefits to complement system health assessments with comparisons between the systems within a fleet already for relatively small fleets, i.e., 5 systems up: In our experiments, which we describe in Section 5 below, we handled variation and noise that were large compared to the absolute value of measurements. Still, the fleet-based health assessment often provided similar observations and cases for insightful comparisons, while the investigation of observations even relied on those to determine its results.

However, these benefits are not free, as the effort to compare a system with other systems increases linear with the number of systems – which we find acceptable. Still, every application will have an optimal fleet-size that ensures (near) optimal diagnosis without undue efforts. This size depends on the variation between systems, but early results indicate that our methods do not depend on large fleets.

5. EXPERIMENTAL EVALUATION

We evaluated our fleet-based system health analysis on the two use-cases introduced above: 1) assessment of a software and its updates, wherein the software is part of a process control loop that corrects physical drift behavior; and 2) the early detection of execution faults. The two-dimensional drift behavior is non-monotone and manifests in an overlay of various frequencies that depend of the origin on the drift.

To illustrate the diagnostic power of our health assessment in a way that we can share, we generated data sets for six systems showing the drift behavior projected to one dimension within shortened time-series of twenty steps (see Appendix). The data sets mirror existing variability within the fleet and the volatility of the processes that we handle.

5.1. Healthy Process Correction

The assessment of software-based process functions like drift-correction works very well:

The monitors' performance is flawless in situations where they check a direct input-output relation between measurements and observable corrective actions, even with noise on both sides and hidden impact factors on the latter: Table 1, showing the health assessment for the twenty time-steps and six systems, holds consistently high health values for all systems and time-steps in which the software worked correctly and immediately detected the wrong software functionality, which started in time-step 6 for systems 3 and 4 and in time-step 11 for systems 5 and 6. (see Appendix, Experiment 1 for further details)

	1	2	3	4	5	6
1	0,87317	0,89580	0,86815	0,88928	0,87714	0,89057
2	0,88707	0,88377	0,87839	0,87681	0,88631	0,89522
3	0,88811	0,89037	0,88422	0,87807	0,88242	0,86758
4	0,87648	0,89124	0,86711	0,86845	0,88390	0,88225
5	0,88678	0,87504	0,88844	0,85863	0,89015	0,88589
6	0,89639	0,87730	0,1	0,1	0,87459	0,88804
7	0,88550	0,88470	0,07	0,07	0,88572	0,87427
8	0,89248	0,88589	0,05	0,05	0,89458	0,89437
9	0,89466	0,87397	0,01	0,01	0,89026	0,88943
10	0,89549	0,88963	0,001	0,001	0,89130	0,89330
11	0,88531	0,86984	0,001	0,001	0,1	0,1
12	0,88321	0,88223	0,001	0,001	0,07	0,07
13	0,89029	0,89073	0,001	0,001	0,05	0,05
14	0,88586	0,88325	0,001	0,001	0,01	0,01
15	0,88152	0,88412	0,001	0,001	0,001	0,001
16	0,87776	0,87328	0,001	0,001	0,001	0,001
17	0,86978	0,87284	0,001	0,001	0,001	0,001
18	0,86513	0,84238	0,001	0,001	0,001	0,001
19	0,89407	0,89436	0,001	0,001	0,001	0,001
20	0,89490	0,88072	0,001	0,001	0,001	0,001

Table 1. Health Assessment Experiment 1.

The same detection power is even available for more difficult assessments, e.g., for computations that take in earlier corrective intent instead of new measurements – a situation that speeds up the corrective loop but amplifies noise. (Experiment 2 in the Appendix.)

While we see that the monitors handle complex situations in a noisy environment, we still saw individual low values for the health of the process correction, indicating uncertainty in the diagnosis. The dedicated analysis of software versions described in Section 4.2, however, leaves no such doubts, even though it is using the same numbers to execute its task: it suspects the software update as being broken (low value) immediately after its introduction in two systems and confirms that suspicion the moment the software is rolled out to additional systems. This is shown in Table 2 in the sequence for the twenty time-steps for the two software versions (SW 1 & 2) for Experiment 1 (top) and 2 (bottom).

	1	…	5	6	7	8	9	10	11	…	20
SW 1	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
SW 2	NaN	NaN	NaN	0,345	0,290	0,273	0,233	0,214	0	0	0

	1	…	5	6	7	8	9	10	11	…	20
SW 1	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
SW 2	NaN	NaN	NaN	0,345	0,290	0,273	0,233	0,214	0	0	0

Table 2. Software Assessment for Experiments 1 & 2.

5.2. Healthy Execution

We generated another data set to investigate the detection of execution errors as laid out in Section 4.3. In this data set, we inserted faults of two different types: capped corrections, i.e., corrective actions limited in their scope due to physical constraints, and corrections that failed to happen (dropped execution). To stress test the health monitoring, we ran this detection as black box: the intended behavior was hidden from the monitors, which therefor could only judge the health of the execution based on past observations within the system and the fleet.

As detailed in Experiment 3 in the Appendix, the health monitoring worked well. We had no false positives and only two false negatives within 360 estimates, both of which were corrected by the detection in the subsequent time-step.

By comparison, the Western Electric Rules break down for this scenario, showing significantly more false positives and hardly any true positives. This is because execution errors like the ones we face are not a symptom of a process running out of control and thus outside the intent of WER.

5.3. Assessment

In total, the specificity and sensitivity of the software and execution investigations with probabilistic health monitors exceeded our high expectations.

Looking into at details of the health assessments, we noticed that the individual parts of the monitoring compensated for each other weaknesses and never struggled collectively. The fleet aspect proved to be especially powerful in this, greatly stabilizing the overall outcome, as it often found comparable situations that led to the correct framing of observations.

6. CONCLUSION

Our work on fleet-based system health assessment centers on two contributions:

First, the improvement of health assessments of individual systems with comparisons to other systems within a fleet of comparable systems. Here, we discussed the difficulty to transfer insights between systems: We argue that many practices especially within the field of remaining useful life predictions assume a similarity between systems that disregards relevant differences in configurations or features, operational modes, or environmental circumstances – but given causal modeling, we are able to reason about such differences, which allows us run sensible comparisons.

Such advanced comparisons to other systems of the fleet then form an integral part of the probabilistic health monitors we introduce, with similarity computations to the system's own past and its intended function as complementary functions. Altogether, we see the three parts of the health monitors to support each other and even seemingly eliminate individual weaknesses, such that the resulting system health assessment reaches top performance.

Second, we embed our probabilistic health monitors within a dynamic probabilistic reasoning process to investigate change within the fleet. This offers an assessment of any intervention, like software upgrades, that are rolled out to the fleet in order to ensure that it improves the fleet's health. Here, the comparison between systems and the reasoning about their differences and how those change over time is essential to the fast and accurate detection of issues.

We see the relevance of both contributions in our work in the high-tech industry, where we observe that industry often must abandon the notion of stable processes. Instead, they handle constantly changing production runs with various interwoven optimization loops. This makes novel process control techniques necessary and the fast and accurate health assessment of processes and their corrective actions that we show is an asset for this.

As the variation and volatility of modern processes that we handle contributed to the fast-growing interest in digital twins within recent years, our techniques need to be compatible to those. While this is not in the focus of this publication, we can point out that the computations we use in our assessments are efficient enough for real-time use on live data within digital twins. This offers the needed elasticity to handle predictive and preventive maintenance, process control and correction that covers even execution faults, as well as the investigation of interventions like software updates or hardware upgrades.

As some of these applications need dedicates analysis models that rely on expert knowledge, we seek progress in the fusion of model-based and data-based techniques in future work.

REFERENCES

Balke, A. & Pearl, J. (1994). Probabilistic evaluation of counterfactual queries. 12th AAAI National Conference on Artificial Intelligence (AAAI'94) (pp.230–237). AAAI.

Barber, D. (2012). Bayesian reasoning and machine learning. Cambridge University Press.

Baru A. (2018). Three Ways to Estimate Remaining Useful Life for Predictive Maintenance. Mathworks. Webpage: www.mathworks.com/company/newsletters/articles/three-ways-to-estimate-remaining-useful-life-for-predictive-maintenance.html. Retrieved: 30.03.2020

Berthelsen E. (2018). Market Trends: Predictive Maintenance Drives IoT in Manufacturing Operations. Gartner Research Report ID: G00350483. Gartner.

Borth M., & van Gerwen, E. (2018). Data-driven Aspects of Engineering. *13th Annual Conference on System of Systems Engineering* (pp. 219–224), Paris. IEEE.

Borth, M., & Barbini, L. (2019). Probabilistic Health and Mission Readiness Assessment at System-Level. *Annual Conference of the PHM Society*, 11(1). doi.org/10.36001/phmconf.2019.v11i1.777

Hardesty, L. (2015). *Probabilistic programming does in 50 lines of code what used to take thousands*. phys.org/MIT. phys.org/news/2015-04-probabilistic-lines-code-thousands.html

Heckerman, D., Mamdani, A., & Wellman, M.P. (1995). Real-world applications of Bayesian networks. *Communications of the ACM*, vol. 38, 3, pp. 24-26. doi=10.1145/203330.203334

Jensen, F.V. (2007). *Bayesian Networks and Decision Graphs.* New York: Springer.

Lerner, U., Parr, R., Koller, D., & Biswas, G. (2000). Bayesian fault detection and diagnosis in dynamic systems. *AAAI Conference on Artificial Intelligence* (pp. 531-537). AAAI Press.

Mulders M. & Haarman, M. (2018). Predictive Maintenance 4.0 Beyond the Hype. pwc and mainnovation.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29 (3): pp. 241-288. doi:10.1016/0004-3702(86)90072-X.

Pearl, J. (2009). Causality. Cambridge University Press.

Pfeffer, A. (2016). Practical probabilistic programming. Manning Publications Co.

Roychoudhury, I., Biswas, G., & Koutsoukos, X. (2008). Comprehensive diagnosis of continuous systems using dynamic Bayes nets. *19th International Workshop on Principles of Diagnosis* (pp. 151-158).

Ricks, B. W., & Mengshoel, O. J. (2009). Methods for probabilistic fault diagnosis: An electrical power system case study. Annual Conference of the Prognostics and Health Management Society (PHM-09).

Xiao-Sheng Si, Wenbin Wang, Chang-Hua Hu, Dong-Hua Zhou (2011). Remaining useful life estimation – A review on the statistical data driven approaches. *European Journal of Operational Research*, Volume 213 (1), pp. 1–14. doi.org/10.1016/j.ejor.2010.11.018

Western Electric Company (1956). Statistical Quality Control Handbook. (1 ed.), Western Electric Co.

Wirth R., & Reinartz, T.P. (1996). Detecting early indicator cars in an automotive database: a multi-strategy approach. *Second International Conference on Knowledge Discovery and Data Mining (KDD'96)* (pp. 76–81). AAAI Press.

APPENDIX

This appendix describes the setup of the experiments that we included to illustrate our work, as described in Section 5. We executed two of these experiments in a grey box and one in a black box setting, such that the health monitors cannot access the ground truth of the investigated functions, as that reflects non-observability of process steps and measurements noise in industrial settings.

In *Experiment 1*, the monitored systems respond directly to drift that the 6 systems measure over 20 time-steps:

	1	2	3	4	5	6
1	0,08318	0,09437	0,08504	0,08767	0,07919	0,08884
2	0,10590	0,43727	0,17113	0,50547	-0,03024	0,22978
3	0,33675	0,85449	0,26117	2,68949	-0,33644	-0,92863
4	0,61562	1,25285	0,32156	4,83051	-0,40802	-1,15255
5	0,85927	1,87032	0,45639	6,01404	-1,30681	-1,34669
6	0,97446	2,54015	0,55257	8,13806	-1,86279	-1,26722
7	0,99826	3,18430	0,67539	8,19158	-2,56524	-1,43377
8	1,23039	3,63903	0,71085	6,84901	-2,88373	-1,76493
9	1,46020	3,37264	0,80857	8,66581	-3,36993	-2,16967
10	1,69834	4,11483	0,80039	9,15564	-3,88939	-1,83336
11	1,82069	3,92779	0,98783	7,94638	-5,04925	-2,91566
12	2,23535	4,67182	1,08078	9,71939	-6,60397	-3,20718
13	2,53602	5,49194	1,14706	11,44663	-6,29128	-3,59570
14	2,93456	5,03960	1,26228	12,85140	-7,23911	-4,70066
15	2,79205	5,89861	1,29053	15,55428	-8,92182	-5,12672
16	3,13616	6,82097	1,43934	17,82150	-9,57781	-5,37022
17	3,50989	6,54178	1,48418	19,17648	-8,72866	-5,03838
18	3,57893	8,44073	1,68970	19,34494	-8,15535	-5,45262
19	4,16325	8,22091	1,63213	18,98093	-9,67009	-5,02907
20	4,31406	7,62859	1,72772	22,93833	-9,52580	-6,23554

As stated, these measurements are noisy. The true physical drift together with the corrections that get applied determine the true physical situation in subsequent time steps, which results in these measurements that are again noisy.

	1	2	3	4	5	6
1	0,01682	0,00563	0,01496	0,01233	0,02081	0,01116
2	0,00669	0,05825	0,02094	0,07625	-0,00525	0,01549
3	0,01942	0,09520	0,03277	0,38572	-0,05388	-0,09436
4	0,07594	0,13448	0,07125	0,49471	-0,03214	-0,06712
5	0,07351	0,25804	0,02961	1,44859	-0,12105	-0,13319
6	0,02402	0,29850	1,14419	17,02940	-0,28578	-0,06117
7	0,11705	0,28096	1,36910	17,22082	-0,32031	-0,28530
8	0,10770	0,35327	1,48921	15,40352	-0,14735	-0,11016
9	0,04870	0,58180	1,68418	18,34033	-0,36299	-0,04701
10	0,10648	0,33160	1,76511	18,93590	-0,36864	-0,15393
11	0,23551	0,84578	2,05710	17,62643	-11,05868	-5,95192
12	0,35121	0,77417	2,24660	20,20541	-13,60144	-6,82132
13	0,29632	0,25751	2,40532	23,98289	-13,59879	-7,56893
14	0,14167	0,92502	2,62025	26,04986	-15,37911	-9,84205
15	0,52216	0,62637	2,74194	30,92965	-18,41339	-10,93697
16	0,40442	0,58252	2,98344	34,66081	-19,53171	-11,18860
17	0,24702	1,06417	3,12251	36,03324	-18,40965	-10,58412
18	0,42352	-0,06160	3,40918	36,13019	-17,07440	-11,03943
19	0,19410	0,42963	3,45134	37,69765	-19,48374	-10,79995
20	0,30916	1,14534	3,61852	39,92176	-19,79175	-12,85005

As the goal of the software is to keep drift as close to 0 as possible, we see some systems drift out of their operational window. Some, however, do not and it is not immediately visible that system 3 developed an issue after a software update in time step 6, as the system is still working well.

Our health monitors assess the systems' behavior accurately and immediately (1 indicating certainty of perfect health, 0 indicating a zero probability of this system working well, with 0.1 and 0.001 being an initial and final lower ceiling):

	1	2	3	4	5	6
1	0,87317	0,89580	0,86815	0,88928	0,87714	0,89057
2	0,88707	0,88377	0,87839	0,87681	0,88631	0,89522
3	0,88811	0,89037	0,88422	0,87807	0,88242	0,86758
4	0,87648	0,89124	0,86711	0,86845	0,88390	0,88225
5	0,88678	0,87504	0,88844	0,85863	0,89015	0,88589
6	0,89639	0,87730	0,1	0,1	0,87459	0,88804
7	0,88550	0,88470	0,07	0,07	0,88572	0,87427
8	0,89248	0,88589	0,05	0,05	0,89458	0,89437
9	0,89466	0,87397	0,01	0,01	0,89026	0,88943
10	0,89549	0,88963	0,001	0,001	0,89130	0,89330
11	0,88531	0,86984	0,001	0,001	0,1	0,1
12	0,88321	0,88223	0,001	0,001	0,07	0,07
13	0,89029	0,89073	0,001	0,001	0,05	0,05
14	0,88586	0,88325	0,001	0,001	0,01	0,01
15	0,88152	0,88412	0,001	0,001	0,001	0,001
16	0,87776	0,87328	0,001	0,001	0,001	0,001
17	0,86978	0,87284	0,001	0,001	0,001	0,001
18	0,86513	0,84238	0,001	0,001	0,001	0,001
19	0,89407	0,89436	0,001	0,001	0,001	0,001
20	0,89490	0,88072	0,001	0,001	0,001	0,001

Tracing a software update that was rolled out to systems 3 and 4 in time step 6 and to systems 5 and 6 in time step 11, we see the dedicated software monitor estimating the likelihood of the initial software working correctly as 1, but the update initially judged at 0.345 – allowing the possibility of a coincidence – followed by a strict 0 in time step 11.

	1	…	5	6	7	8	9	10	11	…	20
SW 1	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
SW 2	NaN	NaN	NaN	0,345	0,290	0,273	0,233	0,214	0	0	0

In *Experiment 2*, the monitored systems respond to drift such that they compute in earlier actions – or their assumptions of those, as software computations and the executed correction may differ due to limits or software issues. This real-world scenario amplifies the effects of noise and errors. The probabilistic health monitors reflect that, but work:

Basic	1	2	3	4	5	6
1	0,68747	0,62217	0,66929	0,85680	0,87714	0,59214
2	0,70650	0,68922	0,73104	0,65712	0,32558	0,59758
3	0,71664	0,70413	0,59740	0,76194	0,65607	0,76265
4	0,77153	0,73946	0,59683	0,76748	0,67692	0,86671
5	0,80690	0,59727	0,59919	0,78173	0,75822	0,81602
6	0,59988	0,59824	0,1	0,1	0,80333	0,84507
7	0,59936	0,59905	0,07	0,07	0,83543	0,73524
8	0,59919	0,59926	0,05	0,05	0,59979	0,59961
9	0,59977	0,59997	0,01	0,01	0,59893	0,59981
10	0,59944	0,59953	0,001	0,001	0,59931	0,59952
11	0,59914	0,59923	0,001	0,001	0,1	0,1
12	0,59827	0,59889	0,001	0,001	0,07	0,07
13	0,59932	0,59982	0,001	0,001	0,05	0,05
14	0,59977	0,59966	0,001	0,001	0,01	0,01
15	0,59932	0,59950	0,001	0,001	0,001	0,001
16	0,59952	0,59944	0,001	0,001	0,001	0,001
17	0,59976	0,59972	0,001	0,001	0,001	0,001
18	0,59961	0,59996	0,001	0,001	0,001	0,001
19	0,59978	0,59992	0,001	0,001	0,001	0,001
20	0,59977	0,59989	0,001	0,001	0,001	0,001

While the individual assessments show mid-range values, the software analysis remains the same and certain:

	1	…	5	6	7	8	9	10	11	…	20
SW 1	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
SW 2	NaN	NaN	NaN	0,345	0,290	0,273	0,233	0,214	0	0	0

In *Experiment 3*, we introduce execution faults in the drift correction. The health monitoring, working in a black box, only sees the measurements in the following table, where a dark red indicates a dropped correction and a light red a cap.

	1	2	3	4	5	6
1	0,01783	0,00577	0,01401	0,01225	0,02037	0,01081
2	0,00753	0,05965	0,02227	0,07435	-0,00534	0,23590
3	0,01910	0,09610	0,03160	0,36406	-0,05512	-0,08536
4	0,06765	0,13191	0,06829	0,48348	-0,03410	-0,06652
5	0,07161	0,24372	0,02739	1,36457	-0,12700	-0,13536
6	0,02474	0,28486	0,62525	0,67993	-0,28911	-0,06208
7	0,11623	0,29061	0,01798	9,42323	-0,32841	-0,28624
8	0,11263	0,33406	0,06788	8,52080	-0,15371	-0,10879
9	0,04923	0,60291	0,06377	0,94361	-0,33585	-0,04847
10	0,10361	0,33898	0,16732	0,63695	-0,35708	-0,14917
11	0,23881	0,73969	0,07764	1,64200	-5,52902	-3,22904
12	0,36761	0,81949	1,27488	0,75030	-0,34796	-3,43495
13	0,31962	0,25497	0,11072	1,06880	-7,34953	-4,20240
14	0,14363	0,93657	0,10343	0,36180	-0,85647	-5,33664
15	0,55806	0,69252	0,15506	0,93725	-9,19886	-6,02903
16	0,40632	0,57197	0,09346	4,99766	-0,37452	-5,99014
17	0,25318	1,17447	0,16235	21,16356	-9,68919	-5,56370
18	0,40608	-0,06150	0,03017	21,43145	-0,74530	-5,69639
19	0,19681	0,45862	0,18655	8,11612	-9,21369	-5,53691
20	0,30489	1,19181	0,16840	9,41702	-0,75963	-6,60188

The dedicated tracking of the health monitoring computes the likelihoods for each fault type, which the following table shows for each system in the order 'ok', 'capped', 'dropped'. The color-coding indicates the assessment's correctness.

	1			2			3			4			5			6		
1	1,00			1,00			1,00			1,00			1,00			1,00		
2	0,95	0,00	0,05	0,96	0,00	0,04	0,96	0,00	0,04	0,95	0,00	0,05	0,95	0,00	0,05	0,00	0,00	1,00
3	0,96	0,00	0,04	0,96	0,00	0,04	0,96	0,00	0,04	0,96	0,00	0,04	1,00	0,00	0,00	1,00	0,00	0,00
4	0,96	0,00	0,04	0,96	0,00	0,04	0,96	0,00	0,04	0,96	0,00	0,04	0,95	0,00	0,05	0,96	0,00	0,04
5	0,96	0,00	0,04	0,96	0,00	0,04	0,96	0,00	0,04	0,96	0,00	0,04	0,96	0,00	0,04	0,96	0,00	0,04
6	0,96	0,00	0,04	0,96	0,00	0,04	0,00	0,00	1,00	0,97	0,00	0,03	0,96	0,00	0,04	0,96	0,00	0,04
7	0,96	0,00	0,04	0,96	0,00	0,04	1,00	0,00	0,00	0,00	0,00	1,00	0,96	0,00	0,04	0,96	0,00	0,04
8	0,96	0,00	0,04	0,96	0,00	0,04	0,96	0,00	0,04	0,46	0,00	0,54	0,96	0,00	0,04	0,96	0,00	0,04
9	0,96	0,00	0,04	0,96	0,00	0,04	0,96	0,00	0,04	1,00	0,00	0,00	0,96	0,00	0,04	0,96	0,00	0,04
10	0,96	0,00	0,04	0,97	0,00	0,03	0,96	0,00	0,04	0,97	0,00	0,03	0,96	0,00	0,04	0,96	0,00	0,04
11	0,96	0,00	0,04	0,97	0,00	0,03	0,97	0,00	0,03	0,97	0,00	0,03	0,00	0,00	1,00	0,00	0,00	1,00
12	1,00	0,00	0,00	1,00	0,00	0,00	0,00	0,00	1,00	0,97	0,00	0,03	1,00	0,00	0,00	0,46	0,00	0,54
13	1,00	0,00	0,00	1,00	0,00	0,00	1,00	0,00	0,00	0,99	0,01	0,00	0,00	0,00	1,00	0,46	0,00	0,54
14	1,00	0,00	0,00	1,00	0,00	0,00	1,00	0,00	0,00	0,99	0,01	0,00	0,99	0,01	0,00	0,46	0,00	0,54
15	1,00	0,00	0,00	1,00	0,00	0,00	1,00	0,00	0,00	0,99	0,01	0,00	0,00	0,00	1,00	0,46	0,00	0,54
16	1,00	0,00	0,00	0,99	0,01	0,00	0,99	0,01	0,00	0,01	0,99	0,00	0,99	0,01	0,00	0,46	0,00	0,54
17	0,99	0,01	0,00	0,99	0,01	0,00	0,99	0,01	0,00	0,00	0,00	1,00	0,00	0,00	1,00	0,46	0,00	0,54
18	0,99	0,01	0,00	0,99	0,01	0,00	0,99	0,01	0,00	0,46	0,00	0,54	0,99	0,01	0,00	0,46	0,00	0,54
19	0,99	0,01	0,00	0,99	0,01	0,00	0,99	0,01	0,00	0,99	0,01	0,00	0,00	0,00	1,00	0,46	0,00	0,54
20	0,99	0,01	0,00	0,99	0,01	0,00	0,99	0,01	0,00	0,01	0,99	0,00	0,99	0,01	0,00	0,46	0,00	0,54

We see only two false negatives that are corrected within one time-step. The observed uncertainty in the diagnosis occurs only within sequences of errors, which are unlikely happenstances that we included as a stress test.