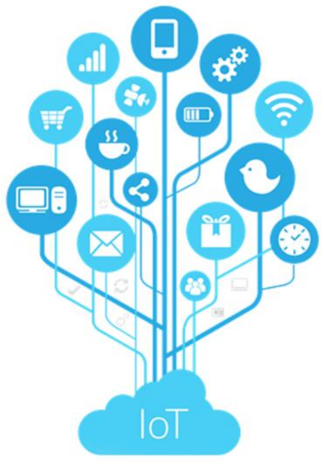
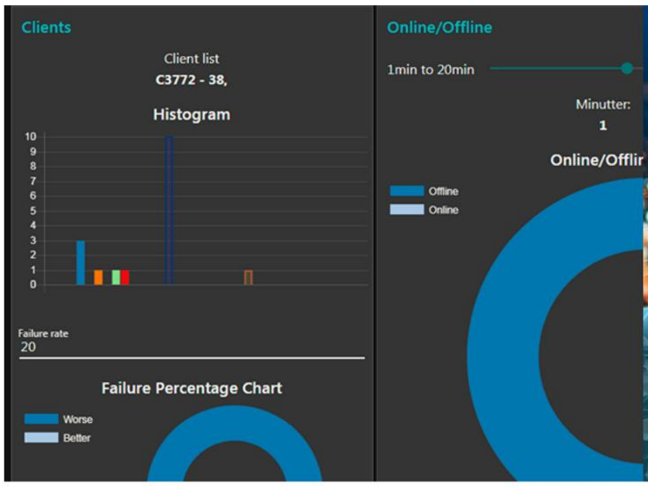
IoT Troubleshooting

WHITEPAPER
2020





FORCE Technology
Venlighedsvej 4
2970 Hørsholm, Danmark
+45 43 25 14 00
+45 43 25 00 10
info@forcetechnology.dk

FORCE Technology Norway AS
Nye Vakås vei 32
1395 Hvalstad, Norge
+47 64 00 35 00
+47 64 00 35 01
info@forcetechnology.no

FORCE Technology
Park Allé 345
2605 Brøndby, Danmark
+45 43 25 00 00
+45 43 25 00 10
info@forcetechnology.dk
www.forcetechnology.com

Title	IoT Troubleshooting Whitepaper
Author	Anders Mynster & Alexandre Alapetite
Task nr.	120-20547.50
Contact person	Anders Mynster
Email	apm@force.dk
Tel.	+ 45 43 25 00 00

Table of Contents

Introduction

There are a plentitude of reports describing why IoT projects fail. Most of these however are concerned about the project management, creation of business concept, change management of the organization as IoT promotes a transition from selling products to delivering an IoT-based service subscription, and so on and so on. This whitepaper however deals with why the technical part of the IoT systems fail – and what to do about it. But the challenges are linked. Because, as your company transitions from a fire-and-forget product to a business model where your earnings are dependent on delivering that service - i.e. everything have to work all the time, the troubleshooting becomes more and more critical. The IoT systems must be operational for as many users as possible, for as much of the time as possible – to meet the SLA (Service Level Agreement) goals and keep subscribers happy.

Some companies, especially the salespeople in those companies, will claim that the system is working all the time. But in reality, that does not mean 100% of the time, at best somewhere in the 99.XX% of the time – sometimes lower. Just to be clear, better than 99,9% means that your system can only have approximately 8.8 hours of system wide downtime per year – including when you are performing software updates maintenance and all that other stuff. When implementing pilots, it was possible to have 100% operational time because it was only for 10 users in a limited time period – but as we as society are implementing IoT systems with thousands of users, we are now facing more operational challenges – and therefore troubleshooting of IoT systems are becoming more and more important.

As the following short list of examples will show, the challenges are:

- interdisciplinary,
- have multiple suppliers as stakeholder,
- have large geographical distribution,
- can be intermittent, slowly increasing, abrupt,
- and dependent on human machine interactions.

Thus, it is essential to have a process that addresses the problem in a data based systematic method, to ensure progress towards the solution, without compromising the need for speed. If a system has broken down, it needs to be fixed asap. Therefore, we have decided to write this guide to how to troubleshoot IoT systems. It is based on +40 years of troubleshooting mechanical and electrical machines and constructions, International standards, and has been upgraded to be most fit for purpose to IoT systems. But before jumping into that we have collected a few examples on where IoT troubleshooting is needed.

- Gateways not collecting data from as many nodes as expected
- Lighting control nodes becoming non-operational in seemingly random patterns
- AC mains powered IoT devices turning off simultaneously
- User interface displays shutting down and becoming permanently damaged
- All remotely monitored and controlled bikes becoming unavailable over night
- Smart garbage cans not transmitting fill level in time for garbage truck calculations
- High number of sensors reporting errors in specific time periods
- Smart building controls opening and closing windows repeatedly
- Sensor data having large fluctuations randomly
- Users abandoning systems in large groups simultaneously

As can be seen, the challenges are many and diverse.

The challenge

IoT-devices are often affected by the installation environment to a much higher degree than traditional IT-systems, as they are distributed throughout the society and IT systems are often installed in a controlled and monitored environment. IoT devices are also more challenged than normal electronics as they devices are often mobile, which means less control of the environment, are low cost and have limited resources with respect to processing power due to battery power constraints. As they are also embedded into normal operation of other systems, such events should also be considered. For instance:

- A temperature monitoring device which is installed in a freezer must also be able to cope with the environment when the freezer is being defrosted, and the data processing must be able to know about this process to not report an anomaly, which might congest the system
- Tracking units for vehicles must be able to tolerate the environment when the vehicle is moving but also the high pressure or chemical cleaning of the vehicle.

Therefore, is it essential to map the events in the physical environment and operation to the data collected by the devices and processing in the server. The monitoring of this is made more difficult than in an IT system, as the processing is distributed between the IoT-devices and the IoT server. Therefore, the data from both the physical world, through observations, measurements, and event mapping, must be compared to the traces found in the data. The troubleshooting process therefore needs to include both specialists in the physical domain and data analysis tools that facilitates the correlation between the physical and virtual world.

Terminology

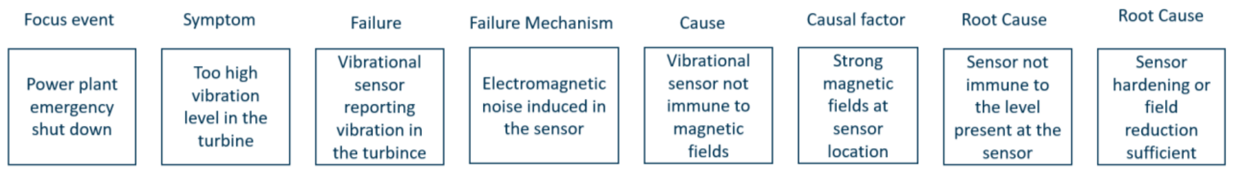
To describe what happens in any system failure it is important to have the right terminology. Sometimes what we experience as "the failure" of the system is only the symptom, and the cause of the failure is hidden deep within the system as a combination of multiple settings and datasets aligning in the wrong way.

- **Focus event**
  Occurrence or change of a particular set of circumstances
- **Symptom**
  The problem experienced by the user or system stakeholder
- **Failure** (or fault of an item) primary and secondary
  Loss of ability to perform as required can be the origin(primary) or a consequence of the first failure(secondary)
- **Failure Mechanism**
  Process that leads to failure
- **Cause**
  Circumstance or set of circumstances that leads to failure or success
- **Causal factor**
  Condition, action event or state that was necessary or contributed to the occurrence of the focus event
- **Root-cause**
  Causal factor with no predecessor that is relevant for the purpose of the analysis
- **Stopping rule**
  Reasoned and explicit means of determining when a causal factor is defined as being a root cause

To demonstrate the use of the terminology is a case where a power plant reactor performed emergency shot downs randomly 1 to 2 times per week. Every time it did so, the power plant operator lost revenue

- Due to loss of production, as it took 4-5 hours to perform a restart
- The turbine lifetime was reduced by 2 weeks for every emergency shutdown, lowering the value of the investment
- Lost reputation as a stable supplier of electricity

The events were caused by a vibrational sensor alarm, that was triggered by a sensor reporting too high vibration levels at the turbine – a safety mechanism to avoid catastrophic failures. The vibrations were not real, but were a result of a magnetic field coupling into the sensor as it was not immune enough towards such a physical environmental factor. Such as:


Focus event
Power plant emergency shut down
Symptom
Too high vibration level in the turbine
Failure
Vibrational sensor reporting vibration in the turbince
Failure Mechanism
Electromagnetic noise induced in the sensor
Cause
Vibrational sensor not immune to magnetic fields
Causal factor
Strong magnetic fields at sensor location
Root Cause
Sensor not immune to the level present at the sensor
Root Cause
Sensor hardening or field reduction sufficient

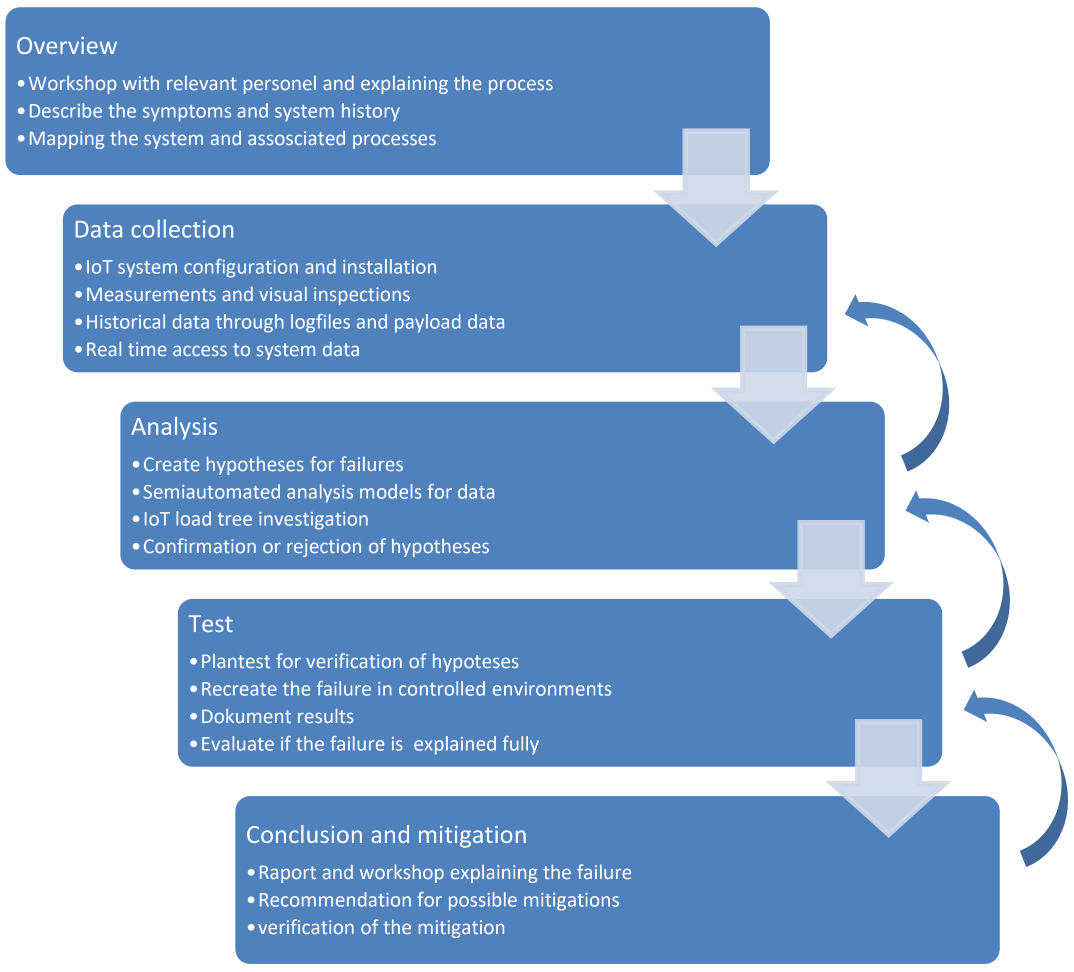
The stopping rule is interesting to discuss here. In this case, the stopping rule was to stop the current series of emergency shutdowns, which required to either make the sensor more immune, or to lower the magnetic fields in the location of the sensor. Stopping rule could have been to avoid making designs with similar events occurring, which would have led to additional actions in the design process where the sensors are chosen, and the physical environment is characterized.

The IoT troubleshooting process

Those familiar with failure analysis will see that there are many similarities between the proposed troubleshooting process and the roots cause analysis (RCA) method. However, RCA is often performed after an accident, in such a case, thoroughness is more important than speed. But when troubleshooting live IoT systems, where users are waiting, time is critical and thus speed must often be considered as a major factor in deciding next steps. It can be compared to first aid for accidents, where the first two steps are to assess the situation and to stop the accident from becoming worse. Similar in IoT troubleshooting, it is important to get an assessment of the situation and stop the problem from becoming worse.

The process is divided into 5 phases. Each consecutive of the other, but in some cases, it is necessary to go back to the previous phase. This often occurs when one hypothesis appears to be the most obvious cause for the event and therefore that hypothesis will be prioritized in the investigation. It is essential for the process to keep all actors as informed as possible about new findings in the individual groups, and that the findings are documented for future reference. When a statement from an actor about the observation is not backed by hard evidence it is essential to be cautious about making conclusions before this documentation is available – but again it is a tradeoff between a fast process and a documentation of the root cause.

The five phases in the troubleshooting process is shown in Figure 1.



Overview
• Workshop with relevant personel and explaining the process
• Describe the symptoms and system history
• Mapping the system and assosciated processes
Data collection
• IoT system configuration and installation
• Measurements and visual inspections
• Historical data through logfiles and payload data
• Real time access to system data
Analysis
• Create hypotheses for failures
• Semiautomated analysis models for data
• IoT load tree investigation
• Confirmation or rejection of hypotheses
Test
• Plantest for verification of hypoteses
• Recreate the failure in controlled environments
• Dokument results
• Evaluate if the failure is explained fully
Conclusion and mitigation
• Raport and workshop explaining the failure
• Recommendation for possible mitigations
• verification of the mitigation

Figure 1 The 5 phases of troubleshooting

Phase 1: Initiation

The Initiation phase is all about getting an overview of the event and the system and to align the participants in the troubleshooting process. What has happened and what are the components involved in the event? Lastly it is essential to be aware that IoT-systems are often developed by multiple companies and actors from the ecosystem. So, in addition to the organizational map it is often beneficial to get an overview of these actors early in the troubleshooting process.

Event characterisation is the first item that should be addressed. Key questions in this phase are:

- Who has observed what?
- When have they observed it?
- Is there documentation of it?
- Are there measurements of it?
    - Where has it been measured?
    - How has it been measured?
- Has anyone else observed it?
- How is the desired state of the system – how the system should work
    - Is it documented in a requirement specification?
        - Were the requirements interpreted correctly by the developers?
        - Were the requirements explained correctly by the client/user?

The second part is mapping the event to a system overview both for functional block, for the system history, and for the development and operational actors involved. Here the key questions are:

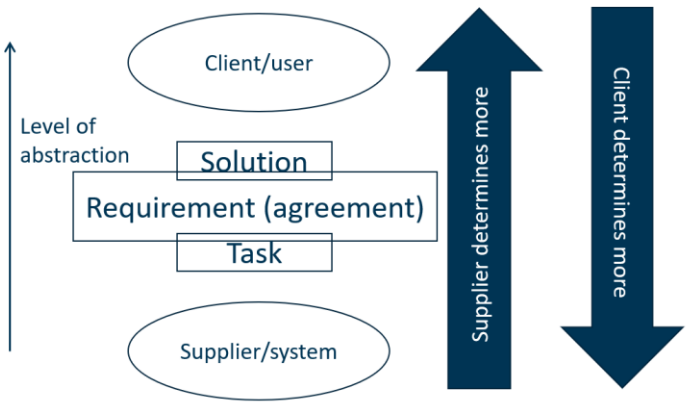
- System overview
    - What are the systems functional modules?
        - Is there a block diagram?
        - Can it be mapped to an IoT reference architecture and IoT domain model?
    - What is the physical and virtual environments of the functional blocks?
    - What are the company and employee processes related to the system?
- System History
    - Has the system been developed according to a standardised process with quality control and tests?
    - Has the problem just occurred or has is gradually become worse?
    - Has design or production changes been made to the system recently?
        - Software
        - Firmware
        - Hardware
- System development and operations actors
    - Who are responsible for developing which parts of the system?
    - Who are responsible for the operations of which parts of the system?

As can be seen from the above. The key is to ask a lot of questions in this phase. Here two techniques come in handy: How vs. Why? And 5y's. In addition to these it is beneficial to work with reference concept model and architecture models for IoT systems to get an overview of the interdependencies in the system.

Tool 1: How vs. Why

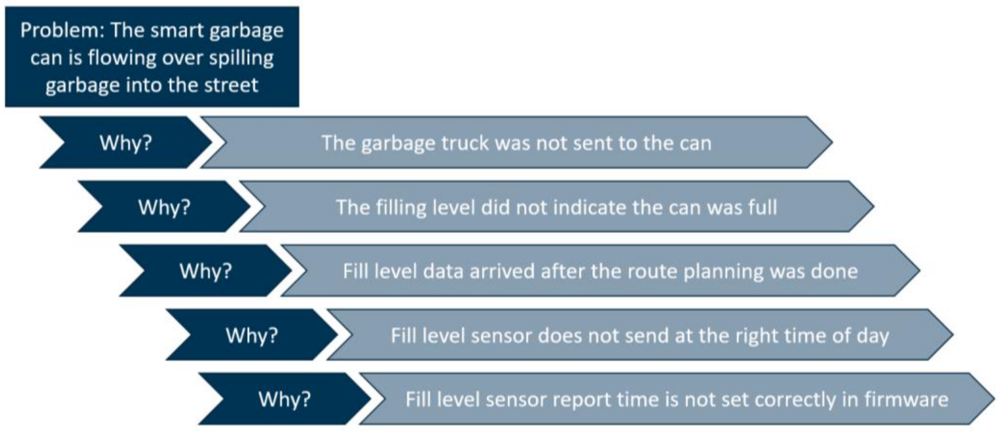
Primarily used for requirements to the system. Here it is beneficial to determine if we want to investigate the implementations or the origin of the requirement from the clients/users

- Ask: ”why?” to move focus towards the client or user
- Ask: ”how?” to move focus toward the supplier or system or way it is implemented



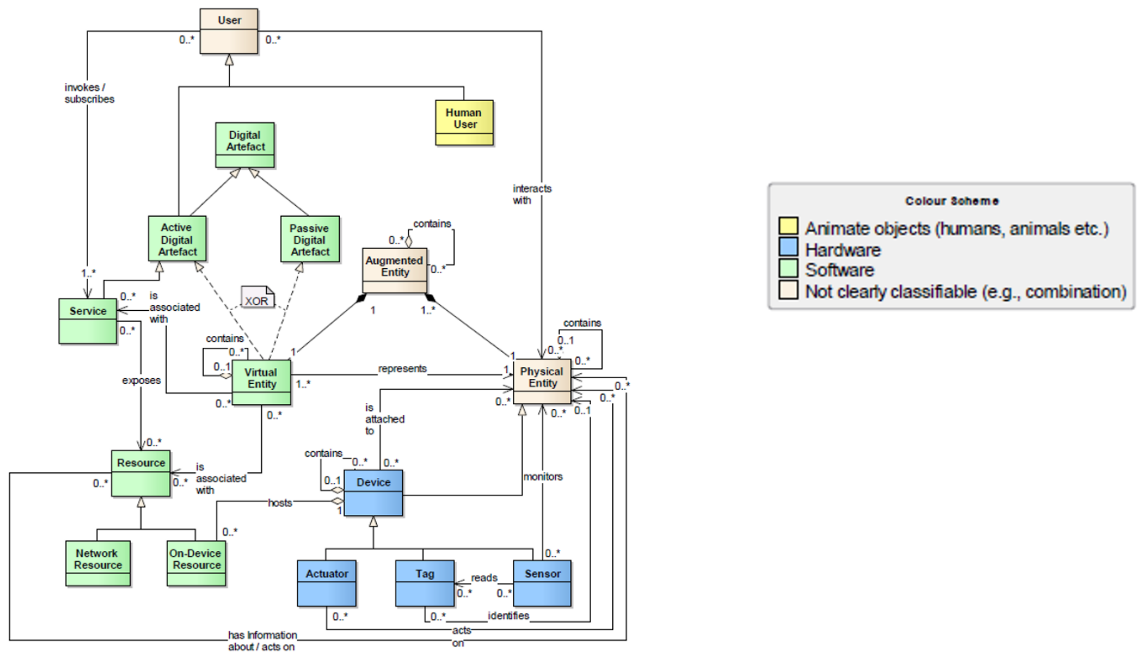
Tool 2: 5y’s

5 y’s or 5 times why, is often used to dig deeper into a problem. It is often easy to describe only the event and its symptoms, and it is important to dig for the cause behind the most obvious causes.


Problem: The smart garbage can is flowing over spilling garbage into the street
Why?
The garbage truck was not sent to the can
Why?
The filling level did not indicate the can was full
Why?
Fill level data arrived after the route planning was done
Why?
Fill level sensor does not send at the right time of day
Why?
Fill level sensor report time is not set correctly in firmware

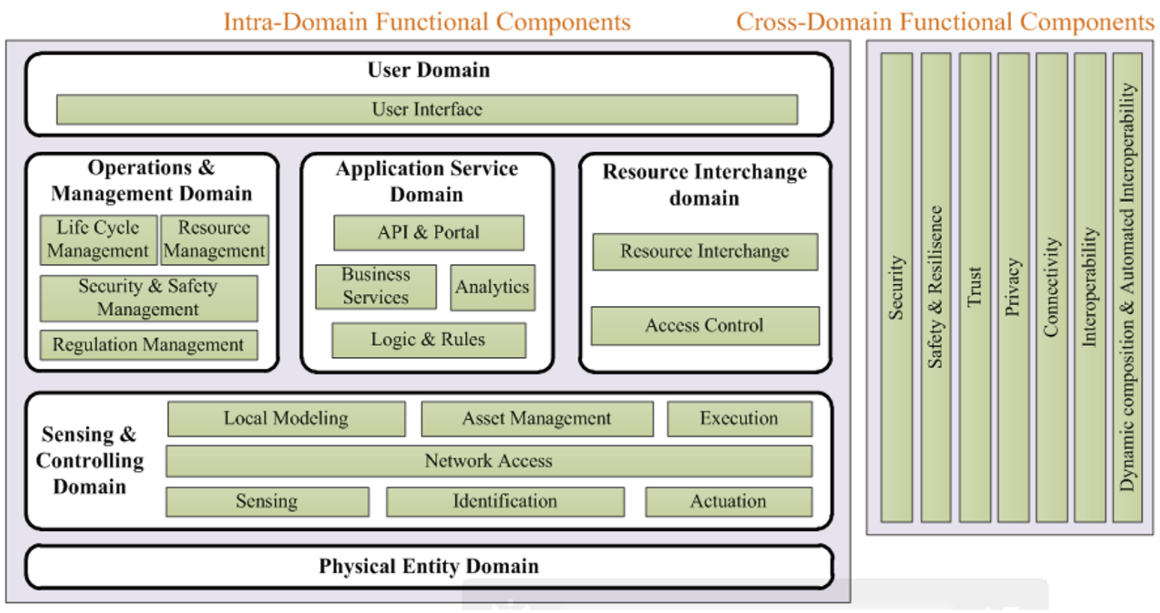
Tool 3: Domain/concept model

The domain model from IoT-A can be used to map the conceptual blocks of the system and how they interact with the users



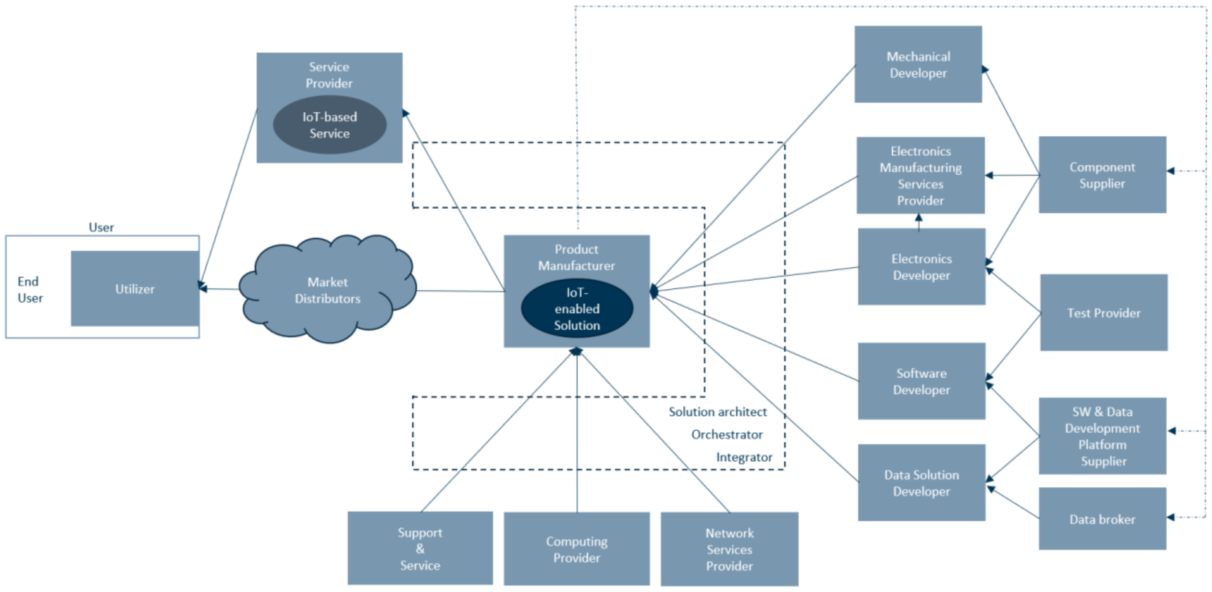
Tool 4: system reference architecture

The system reference architecture from the standard ISO/IEC 30141 can be used to map the system functional blocks of the IoT system, to get an overview of how the system is technically constructed.



Tool 5: Role based IoT-ecosystem mapping

For each of the roles, mark down who is responsible for the development and operation of the system. This includes internal organizational units and external developers or suppliers of standard systems. The mapping should have a contact person for each organization.



Phase 2: Data collection

The data collection phase is essential to collect an evidence base for the hypothesis about the cause for the event and failure. It will serve as a foundation for the analysis in phase three. The first step in the data collection phase is visual inspections of the IoT-devices, both on the hardware in the lab and on installations, here it is essential to get photo documentation on the observations. As I might be relevant for other team members to access this information and for following documentation. Sometimes visual analysis must be complemented on measurements. It is important here not to confuse such data collection with tests, as the measurements are only to collect data and the tests are to provoke the event, failure, or cause. Such measurements can be inspections with microscopes, electrical measurements, or mechanical measurements. Another important action here is to confirm observations discussed in the initiation through interviews, with others that might also have observed, or not observed, the symptoms of the failure.

The second step of the data collection phase is retrieving data from and about the IoT system itself. Here it is essential to get a copy of historical data either through files or APIs, especially during the event, but also in periods where the event did not occur, to establish a baseline for when the IoT system is in normal operation. Lastly, a live data connection should be established to the system, this can either be as

- an end-point clone, which copies all data from the IoT-devices and users to a secondary analysis system,
- by setting up additional gateways, to collect data from the network directly
- by accessing APIs, which is particularly important when trying to troubleshoot failures originating in the data cleaning and decoding process
- by accessing user interfaces, which are often graphical, so here screenshots must be taken to document the data outputs.

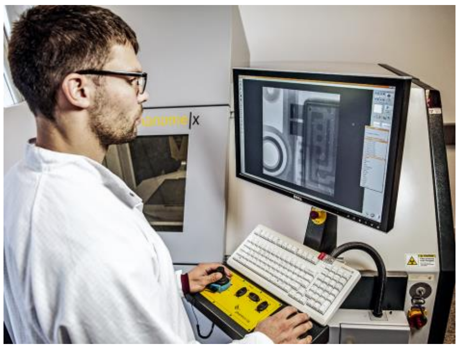
It is important not to engage too much in the analysis phase at this point, but a cross check of data across the system should be performed at an overall level at this stage. I.e. comparison of Network data, raw data, refined data, data products and user interface similarity.

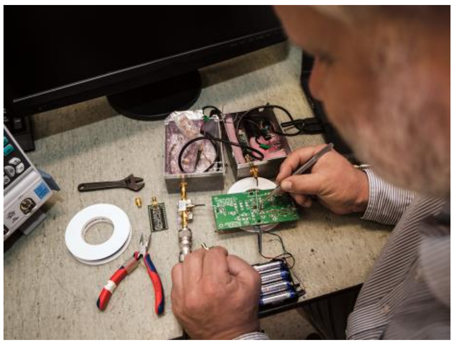
The types of data that should be collected from and about the IoT system are typically:

- Log files and system monitering logs
  - IoT-devices and servers
  - System load
  - Configuration files
  - Software updates
- Time series data
  - Data measured by the devices: Temperature, humidity, vibrations, location, rotational speed, …
  - Data about devices: Remaining battery power, received wireless signals, number of event triggers, etc.

Tool 6: Physical data collection

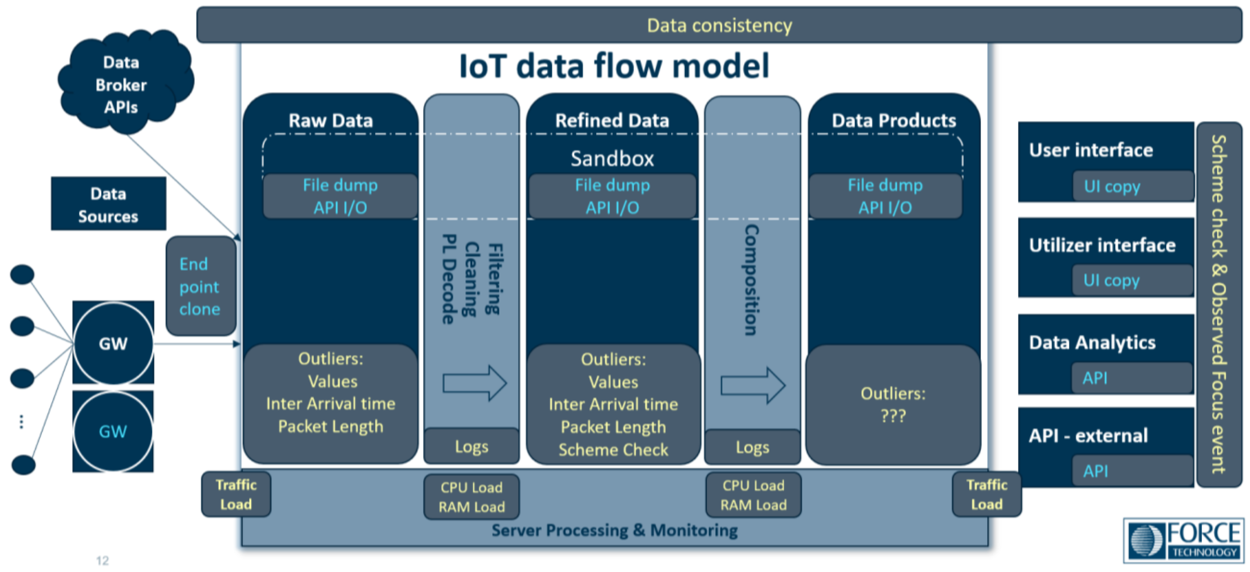
For effective data collection it is essential to have a large toolbox of physical test tools, ranging from microscopes, oscilloscopes with current, voltage and power probes, spectrum analyzers, and network analyzers. Although not a test tool as such, the art of the effective interview should also be part of the troubleshooting toolbox for collecting data from other stakeholders' observations of the event.





Tool 7: Data import for troubleshooting

The essential part of a data import tool is to be able to handle multiple data sources for the system and for various data types such as log files and time series data. Most essential is the interfacing to the typical IoT protocols such as MQTT, HTTPs and REST APIs. The Nordic IoT Centre uses a self-developed toolbox based on opensource components



Phase 3: Analysis

The analysis phase will focus on describing possible failures, failure mechanisms and causes for the event and symptoms. I.e. to formulate a set of hypotheses that can explain what has gone wrong in the system. Following the formulation of the hypotheses, the first step should be to investigate the data collected to create preliminary conclusions based on the documentation available. This can be performed with an event focus in an events and causal factor analysis or the fishbone technique coupled with the items in the load tree from the IoT trust assurance guideline. It is essential to keep track of whether events, causes and failure mechanisms are speculative or based on data. Therefore, a good working practice is to use dashed lines for speculative blocks in the analysis and solid lines for those backed by data. Build a pros and cons list for the individual modules (software or hardware) to investigate what should be tested in phase 4.

An essential aspect in IoT is to validate if the data is compliant with the data description models that are predefined. This can be performed as data validation against Schemas for XML or JSON, and descriptions of the data found in the API – if such a description is missing, that is an indication of cause also.

It is important to remember that sometimes the fault does not necessarily lie in the system itself, but often in the related processes of the system. That is:

- Development of the system
- Storage of IoT devices before installation
- Installation process and location
- Modification to the system (documented or undocumented)
- Runtime environment on both server and IoT devices
- Operational procedures for using the system and accessing data
- Intended versus actual use

If the analysis of the individual modules in the system does not result in one or more plausible hypotheses an additional analysis on data to find possible correlation to activity patterns in other parts of the system should be performed. This is typically called a Swiss cheese problem where multiple events and causes must be aligned simultaneously for the fault to occur.

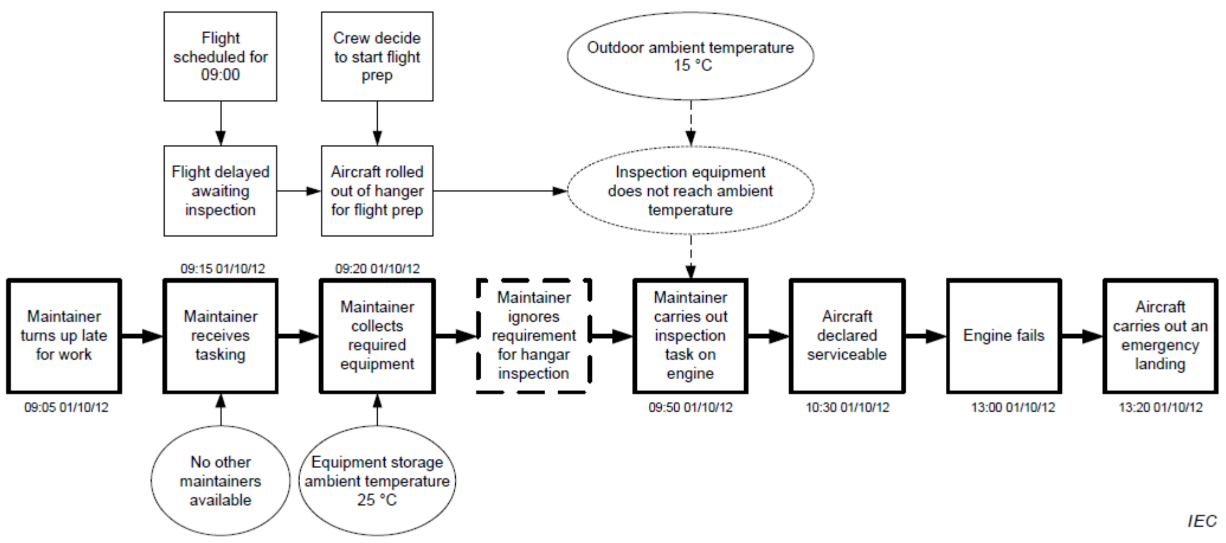
In this phase it is essential to include experts in the system implementation, the system technology, and the application domain.

- Application domain such as public transport, brewing process, wind turbine operations, municipal operations
- Technical sub-components such as wireless communication, mechanical, software, data solutions
- Manufacturers and developers of the sub-components

Tool 8: ECF analysis

The Event and causal factor analysis depict events on a timeline as squares. The conditions for the event to occur is depicted as ovals. If the events or conditions are based on assumptions, they should be outlined as squares and if it is based on data evidence it is outlined by a solid line. One important observation is that multiple event branches can be used if there is suspicion about multiple events and causal factors necessary for the focus event to occur.

IEC 62740:2015 © IEC 2015 – 31 –

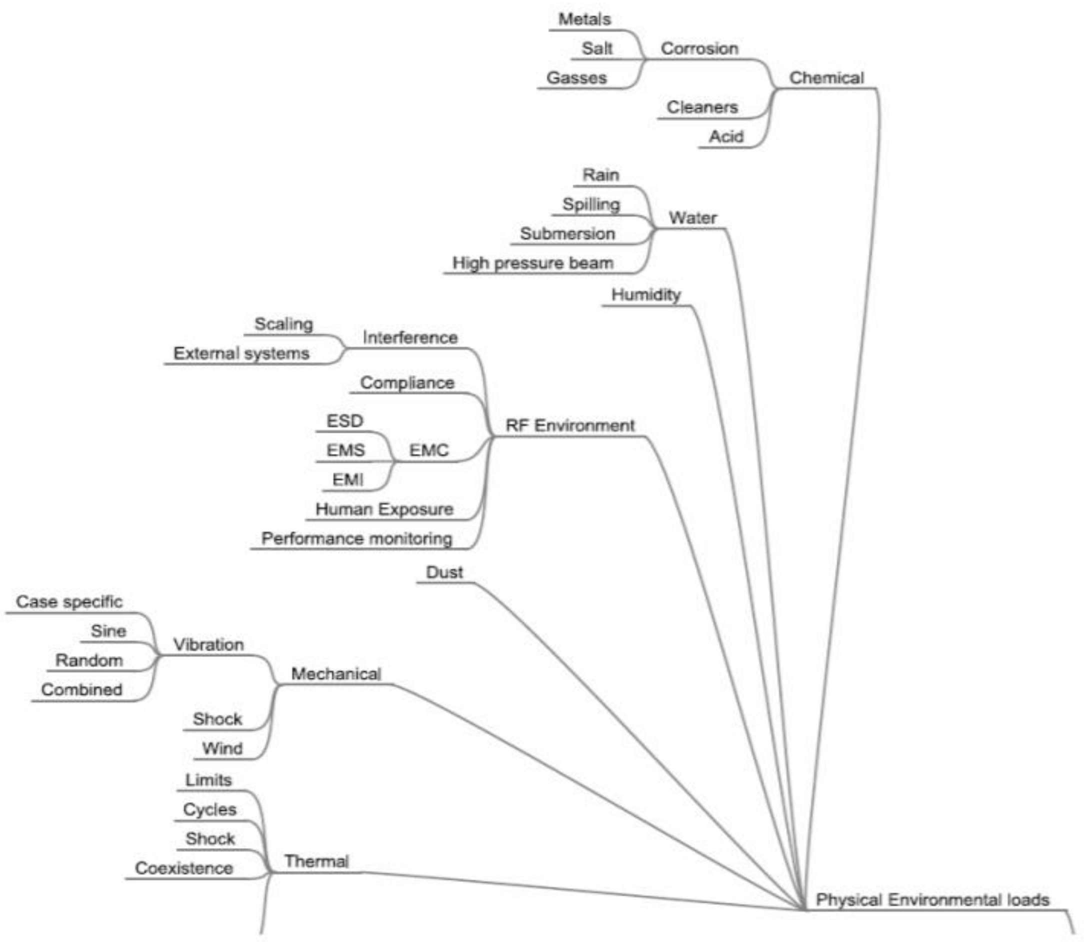

Flight scheduled for 09:00
Crew decide to start flight prep
Outdoor ambient temperature 15 °C
Flight delayed awaiting inspection
Aircraft rolled out of hanger for flight prep
Inspection equipment does not reach ambient temperature
09:15 01/10/12
09:20 01/10/12
Maintainer turns up late for work
Maintainer receives tasking
Maintainer collects required equipment
Maintainer ignores requirement for hangar inspection
Maintainer carries out inspection task on engine
Aircraft declared serviceable
Engine fails
Aircraft carries out an emergency landing
09:05 01/10/12
09:50 01/10/12
10:30 01/10/12
13:00 01/10/12
13:20 01/10/12
No other maintainers available
Equipment storage ambient temperature 25 °C
IEC

Tool 9: IoT load tree for IoT trust assurance

The load tree describes typical mistakes seen in the design phase and operation of IoT systems. It expands the causal factors into 5 primary groups where 4 are relevant for reliability of IoT systems:

- Physical environmental loads
- Physical functional loads
- Virtual environmental loads
- Virtual functional loads

In the picture below a snapshot is taken of the load tree for the physical environment. The Process for use can be standalone but is best suited for analysis in combination with the fishbone technique. Here the 4 groups are inserted as categories in the fishbone analysis. Then for each of the subgroups it should be considered if it can be a causal factor and evaluate the likelihood



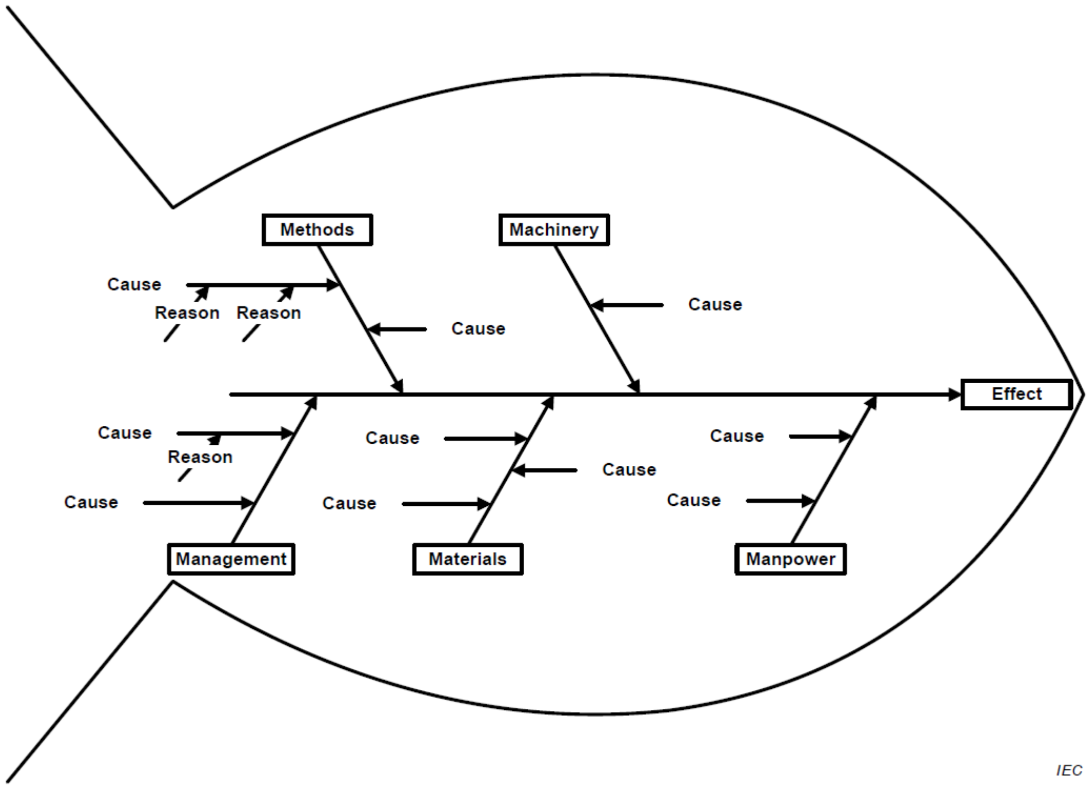
Tool 10: fishbone (Ishikawa) analysis

Start by identifying the focus event. Establish main categories of causes, use the IoT specific causes such as those listed in the IoT load tree or use one of the 4 traditional sets:

- 5Ms: Methods, Machinery, Management, Materials, Manpower
- 4Ps: Place, Procedures, People, Policies
- 4Ss: Surrounding, Suppliers, Systems, Skills

Identify possible causal factors in each category and analyze diagram with respect to:

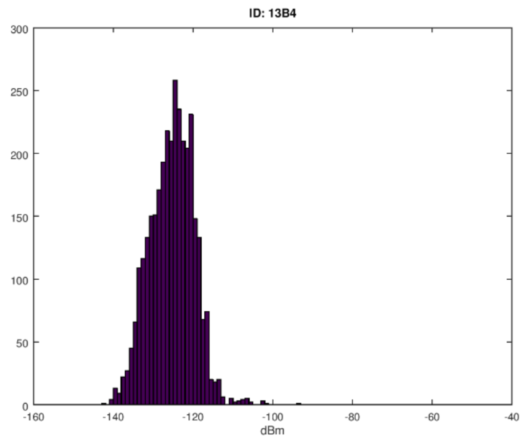
- Balance the diagram to avoid too much focus in one category
- Identify repeated causes – these may be root causes
- Measure to we confirm or reject the cause and to quantify the effect
- Highlight causal factors for actions

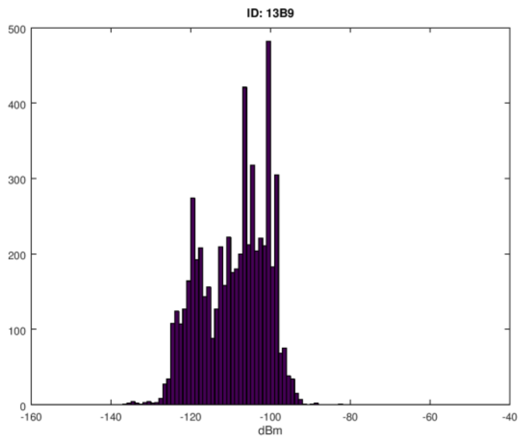


Tool 11: Statistical and timeseries Analysis

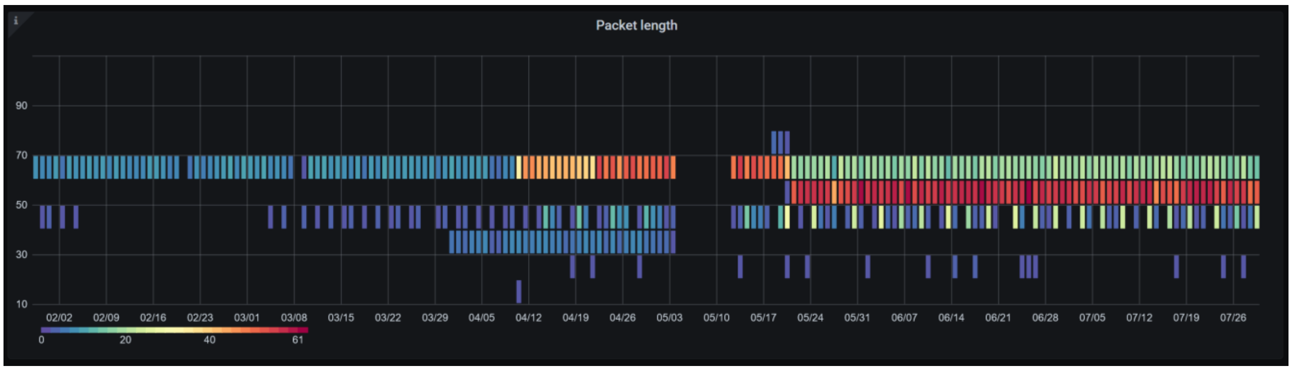
The statistical analysis should be performed to identify outliers and anomaly detections in the data values, interarrival times of packages and traffic loads on the IoT system. The analysis can sometimes result in observations and evidence from a single data source by itself, sometimes it requires combinatorial analysis that looks for correlations between data and use from multiple sources. The analysis levels are typically:

- Simplest: Single source and manual comparison
    - Mean and standard deviation analysis on simple statistical models with being the point of observation
- Advanced: Multisource statistical analysis where data source correlations are known or expected
    - Multivariate analysis using K-means clustering
- Most advanced: Multisource analysis where the data source correlations are unknown
    - Machine learning and neural network training for prediction models including external parameters such as calendar, yearly season, etc.





The timeseries analysis should be based on getting an overview of where to look for correlations. This can for instance be on time distribution of the packet sizes across the system, the inter arrival times of the packets, the amounts of user requests for data, and on the payload data values flowing into the system,



Phase 4: Test and replication

Based on the hypothesis, test should recreate conditions in a controlled manner to reproduce the error to confirm or reject the hypothesis. When planning the tests, it should be evaluated what tests are cheap and fast to perform. Even if it is not the most likely cause of the failure, it is beneficial to exclude quick tests before engaging in larger test programs. The cheapest and fastest first, then second cheapest and fastest second, and so on.

The tests are performed to:

- Reconfirm observations that are not data supported
- Create additional data in case of inconclusive decisions due to lack of data
- document the environment and operational state during the failure

It is important to thoroughly plan the test

- To ensure that the test is evaluating all and only the parameters in the test
- To bring everything to the test site of the physical device
- The tests can be performed in the server sandbox environment
- Test plan documentation can be used to share the plan with all parties involved such that prior to the test the outcome is accepted as valid.

Exposure parameters could be (list not exhaustive – see load tree for IoT systems)

- Physical environment: Humidity, temperature, vibration, electrical fields, electrical pulses, radio interference, faulty installation
- Virtual environment: system load, processing load, cybersecurity attacks, data injection

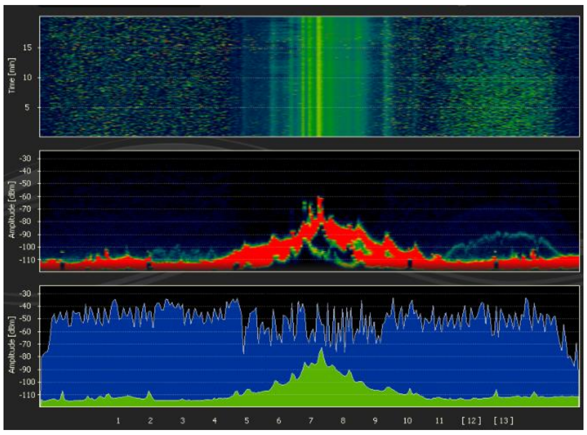
Although documentation is important in all phases of the troubleshooting, it is particularly important in this phase as it will be the data foundation for the conclusion of the troubleshooting and the input on how to mitigate the IoT system. The documentation will also be used for evaluation if the test has conclusive results based on data – i.e. that the stopping rule has been met, such that the troubleshooting process has ended in.

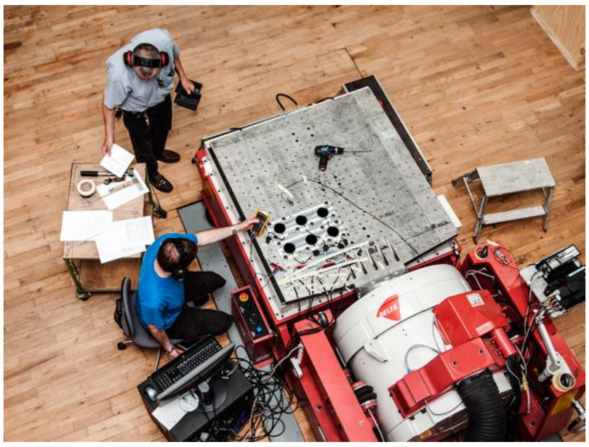
Tool 12: physical environment and user exposure test

The tests are based on the categories in the IoT load tree, but it is important to have a large set of physical exposure facilities available. A separate activity which is essential in some systems is to create roleplaying activities to test the interactions between the users and the system.

The exposures include but is not limited to:

- Vibration
- Shocks and bumps
- Temperature
- Humidity
- Radio interference
- Electromagnetic fields
- User interactions





Tool 13: Code Walkthrough

Code walkthrough is particularly useful for troubleshooting embedded systems. It is a collaboration between the developers and external reviewers. The focus should be on the fastest takeaways from the analysis where no session may take more than 45 minutes before making a break. The external review team is composed to cover knowledge relevant to the task and consists of very experienced people within:

- Embedded software and firmware for Distributed systems
- Specific programming language, protocols, and platform specialists
- Cyber security specialists

The process is the following

- Step 1: The developers explain the code to the reviewers
  - This should be compared to the description of the system described in phase 1 of the troubleshooting process
- Step 2: The reviewers ask questions and "directs" the focus, where to look in the code such as
  - Explain the architecture of the entire program and where the functional blocks are located?
  - Where is this variable initialized?
  - What happens when this queue is empty?
  - Where are interrupts enabled?
  - How was this feature tested?

Findings can result in clarifying additional required physical or software tests or other work, such as statistical data analysis and overview of errors, to be done to strengthen or weaken a hypothesis

- Findings are different e.g.
  - Bugs that can be corrected, e.g. synchronization problems and pointers with wrong end points
  - "unhealthy" software quality, e.g. an overcomplicated solution where a simple alternative is possible
  - A rejection of a hypothesis, e.g. a proposed failure mechanism
  - Sometimes derived actions are needed (e.g. new ways to specify interfaces, or refactoring of the software)
  - The hardware simply hasn't got the capacity to perform the task

Tool 14: Semi-automated software testing

Especially when the code base is large, it is not realistic to manually review every single line of the source code. And even when the code base is not especially large, some types of errors are not easy to spot at a first glance, even within the common types of errors, and especially when it requires an understanding of the logic and data flows between multiple functions.

For those different reasons, it is highly desirable to use some semi-automated methods to flag potential software bugs in a scalable manner.

While some computer languages are much more prone to bugs than others (e.g. C++ being well-known for being bug-prone, while e.g. C# or even more Ada / SPARK are much safer), all languages can benefit from semi-automated source code testing.

One of the main families of methods is "static code analysis", which checks the software without executing it. It can either be at source code level, or computer code level (compiled binary). The simplest tools are "linters", which are able to flag a number of potentially problematic patterns in the source code, for instance `` `if (a = b)` `` which is probably a typo resulting in an assignment instead of an equality test `` `if (a == b)` ``. Linters can also be used to search for problems such as hard-coded values, plain-text passwords, and other common bad practices.

Some other tools in the "code metrics" family are providing statistics such as an assessment of the code complexity from the different portions of the code base, making it easier to prioritise the areas of the code to improve, or in which to search for bugs.

More advanced static code analysis tools can understand the software flow and check for more complicated errors such as null pointers, various leaks, buffer overflows, SQL injections, etc. Some can also perform a "taint analysis" to make sure that trusted and untrusted data do not come in contact with each other.

Those tools not only help with troubleshooting, they also contribute to increasing the software code quality in general and can be advantageously incorporated in continuous development / continuous integration workflows. It is thus important to have a few of those semi-automated software testing tools at disposal. Choosing the proper ones depends on the software language and architecture. However, those tools are not a silver bullet, and are not sufficient by themselves.

Tool 15: Software testing

Software test can be designed to provoke certain types of defects in the code to be activated. In troubleshooting a system that is fully or partly controlled by software, a software test can be designed to have a high probability to reveal certain types of errors (related to the type of test used but potentially also of other types).

- Testing should be performed with different types and different levels
  - Types: Performance tests, Load tests, Functional tests (including malformed inputs, with methods such as "fuzzy testing"), and Usability tests
  - Levels: System test, integration tests, and Unit test
  - Techniques: White box and black box
- Combination with troubleshoot code walkthrough
  - A software test design is best made in combination with a troubleshoot code walkthrough
  - Designing a software test often reveals ambiguities in the requirements, in the architecture or even key stakeholders understanding of the system
- First step
  - Planning the test with regards to types, levels, and technique.
  - What to focus on
  - Who is doing what
- Iteration
  - Following the plan
  - Design individual tests
  - Register findings during the test design (surprisingly enough – there are findings during the design)
- Implement the test

Phase 5: Conclusion and mitigation

The first part of phase 5 is to create documentation of what the troubleshooting process has investigated. The documentation should in particular focus on the Executive summary for the problem and a presentation to the stakeholders to the focus event. This should be used for a resolution workshop with the IoT solution owner and sub suppliers or partners. It is important that all stakeholders are aligned with the conclusions of the process and to facilitate an open discussion.

Second step is to create plan for possible mitigating measures this can include:

- Revised operation procedures
- Altered design for hardware, software, communication, data processing

In rare cases the report can become the documentation used for resolving damage claims between the party who is responsible for the event to occur and the party who has lost operating revenue, loss of reputation etc.