Are your interfaces used as expected?

Runtime Data Analysis
with EB solys

Torsten Mosis, Software Architect, systemticks GmbH
October 25th, 2018, Edinburgh
Tracing Summit

About me

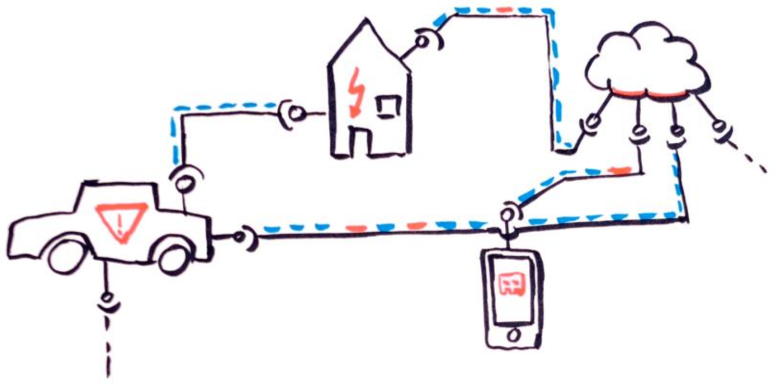
- Co-founder of systemticks (October 2018)
- Product owner of EB solys at Elektrobit Automotive (subsidiary of Continental)
- Software architect at Harman International
- Software developer Siemens VDO

Agenda

- Motivation
    - Increasing complexity in software systems
    - Difficulties in defect location and trouble shooting
- EB solys
    - Architecture & eco system
    - Download
    - Methodology
    - Customization & Extension
- Demo

Motivation

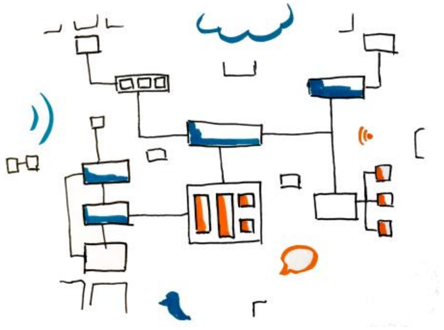
Architecture in future software projects



Current market trends show that software systems in domains such as automotive, IoT or Industry 4.0 are moving into the direction of **service-oriented architecture** on **distributed systems** over **multiple control units** and **devices**.

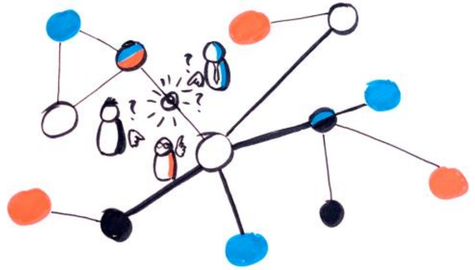
… leads to an increasing complexity

Technical Complexity



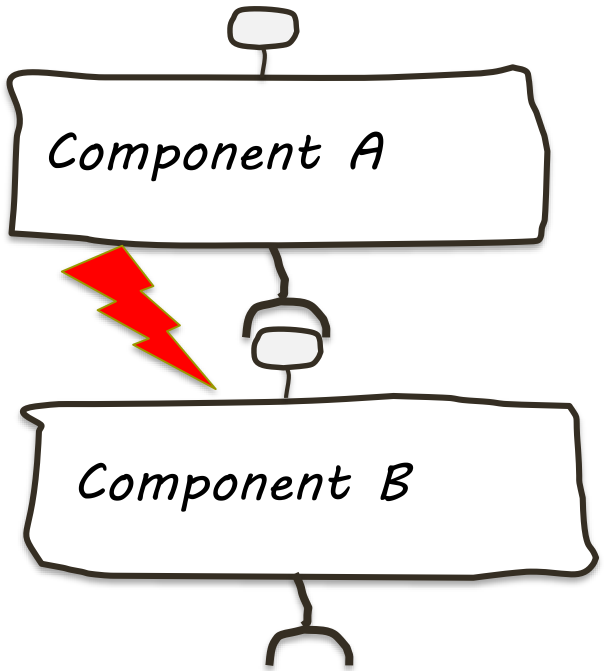
- Number of interacting software components, nodes and partitions
- Usage of remote services in the cloud
- Deployment on multiple devices
- Multiple co-existing programming languages, frameworks, protocols and operating systems

Organizational Complexity



- Different companies and responsibilities
- Different skill and knowledge level
- Multiple countries, time-zones and cultures
- Various set of tools and methodologies

… with inevitable integration issues



What happens:
Although all components have been tested carefully (possibly test-driven) the software runs into trouble when constructing the single components into a larger system.

Typical errors:

- Calls in the wrong order/sequence
- Pre-conditions were not fulfilled
- Post-conditions were not fulfilled
- Calls with wrong range of values
- Call causes performance drawback
- Service called by not-authorized client
- …

… and to expensive actions

Errors and **shortcomings** in such complex systems become **difficult** to **isolate**, since the features are implemented in a **distributed** manner, by cross-cutting multiple layers, services nodes and technologies by **different parties** and **suppliers**.

Solving those errors usually leads to **finger pointing** and actions like **ramping-up** a **huge** testing **infrastructure** and **personnel**, when lacking

- a shared system understanding
- common analysis methodologies
- a consistent tooling

This is **expensive** and **frustrating**.

… requires a consistent analysis solution

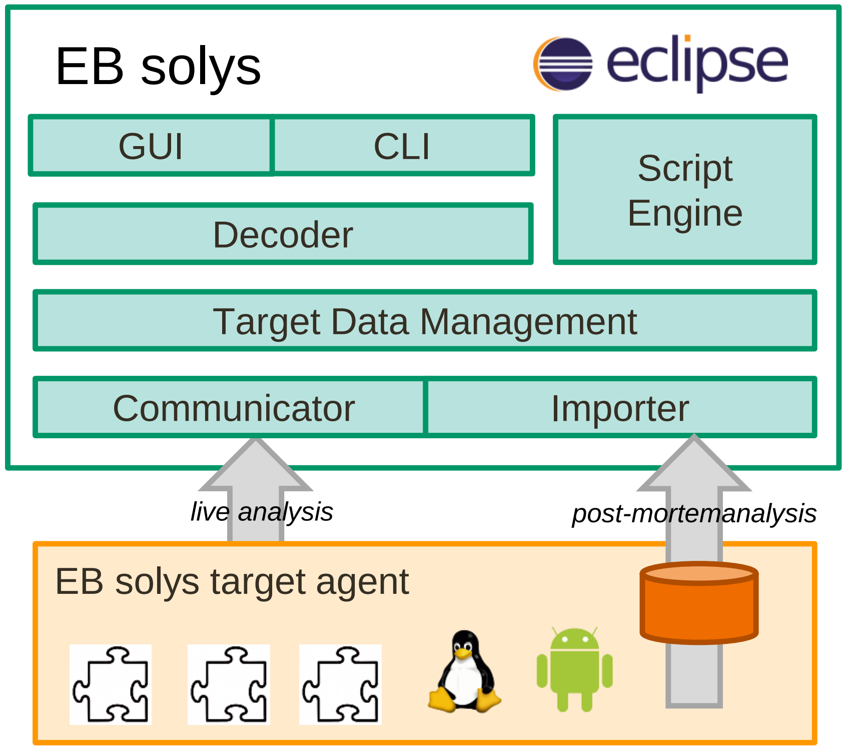
composed of two interconnected approaches:

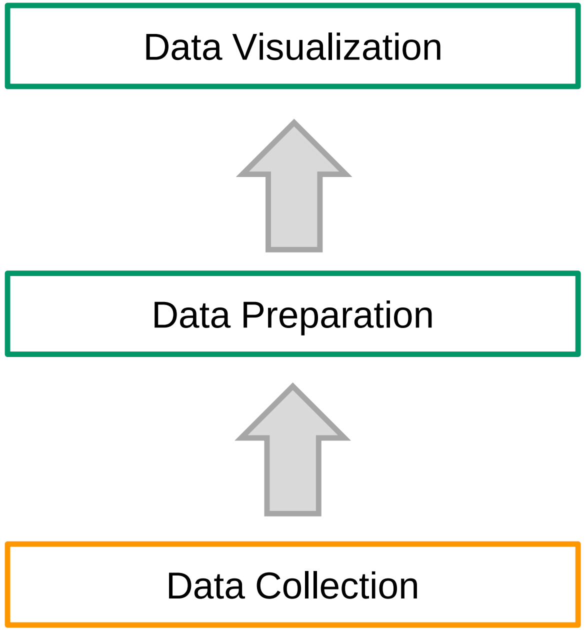
1. applying **methodologies** and **techniques** to your system, to make it **traceable** and **analyzable**
2. developing an efficient **toolchain** with the focus on gathering **valuable** runtime **data** from **different sources** and setting them **in relation to each other.**

This enables you creating a **joint system understanding** and **isolating errors** with significant **less workforce** and gains a **greater insight** into its **operational activity**.

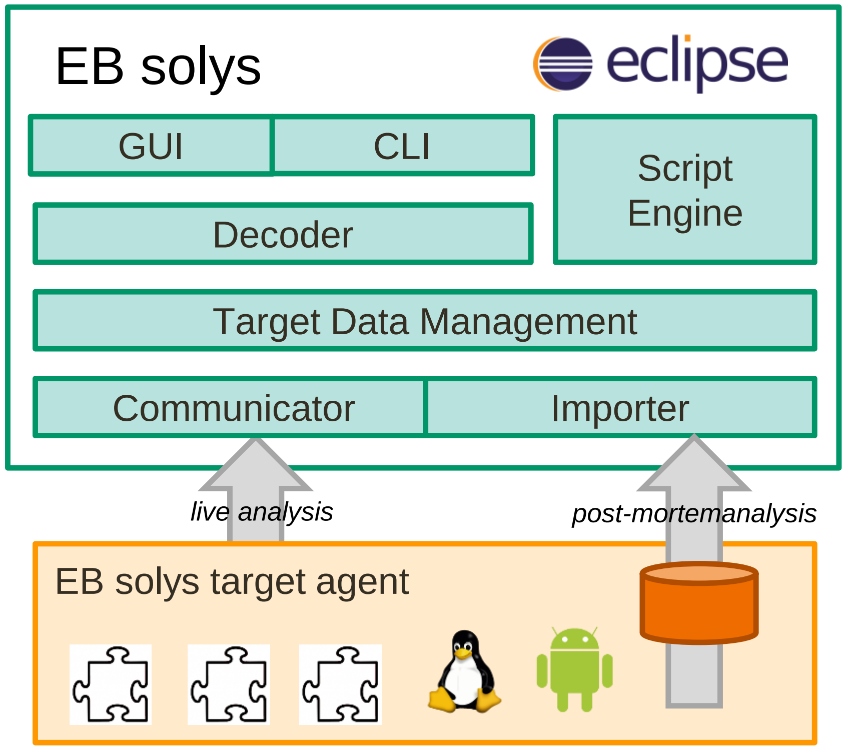
EB solys

EB solys architecture





EB solys on GitHub

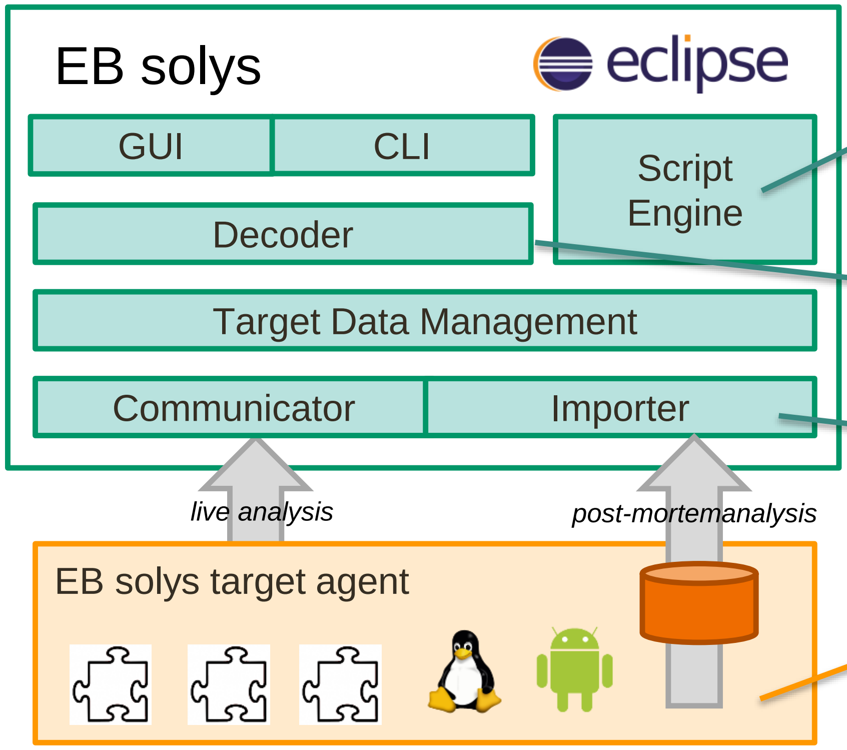


https://github.com/Elektrobit/eb-solys

https://github.com/Elektrobit/eb-solys-target-agent

https://github.com/Elektrobit/eb-solys-android-agent

EB solys customization & extension points



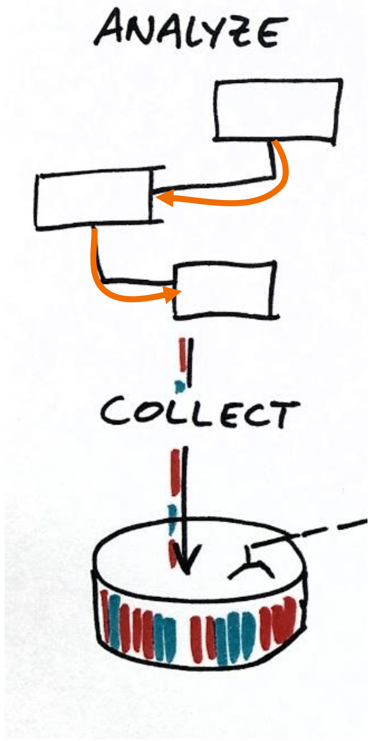
Use the Built-in **Script** Engine to add new functionality by accessing the EB solys raw data and resources

Hook in your own **Decoder**, that transforms arbitrary non-primitive data (e.g. binary payload) into structured readable text

Provide your own **Importer** for any kind of logs, which are not created originally by the target agent

Implement your own **Target Agent Plug-In**, that retrieves project/system specific data

Monitoring communication flow

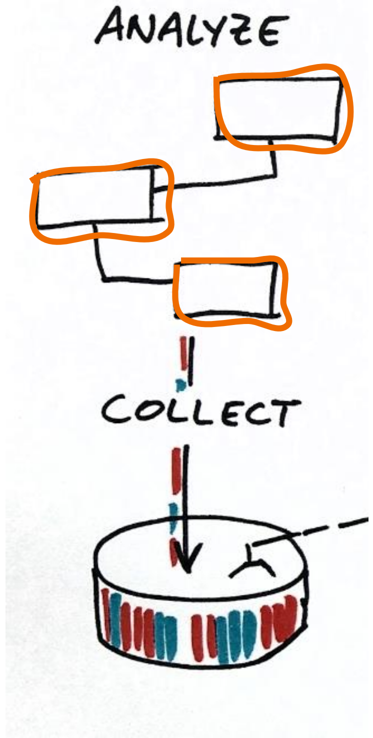


Capturing the **communication flow** of your system, like monitoring

- remote procedure calls
- messages
- events
- broadcasts
- etc.

tells you how your components **interact** with each other, thus reveals its **dynamic behavior** and your **interface design**.

Structured application logging

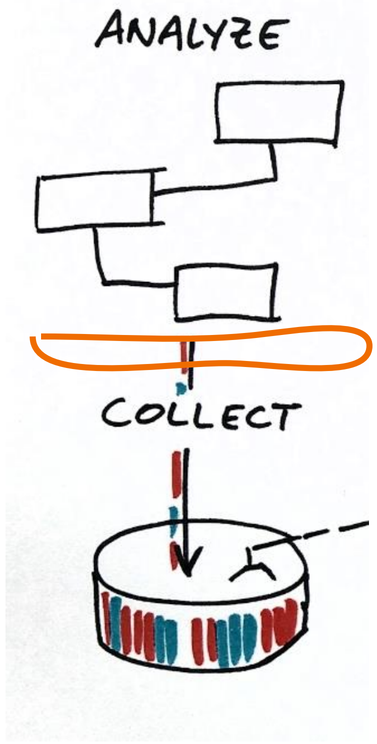


Applying **structured logging** methodologies, like

- using a **consistent**, **predetermined** and **machine-readable** message format
- utilizing **model-driven** approaches
- collecting **semantics** information
- determine appropriate log levels for certain use-cases
- tracing **cross-cutting** features

allows **tracing-back crucial use-cases**, such as startup, shutdown, essential service calls or other high-level functionality.

Capturing resource data

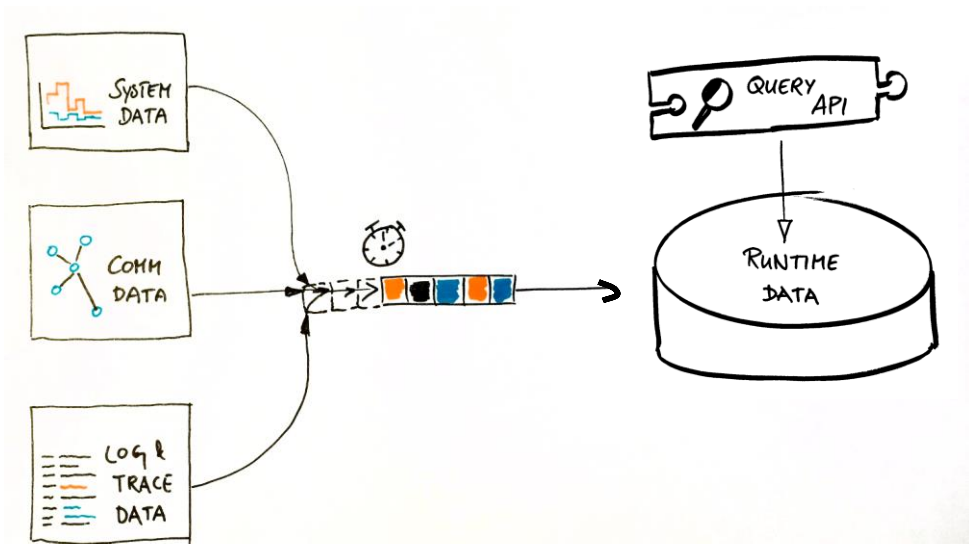


Acquiring data on **system level**, such as

- CPU load
- memory consumption
- file I/O
- network I/O
- etc.

let you draw conclusions from your **non-functional aspects** of your software, such as **workload**, **balance** and **throughput**.

Data formatting and preparation



On top of the data storage we provide a **powerful API** for the purposes of:

- filtering
- searching
- aggregating
- decoding
- correlating
- transforming
- automating

across different **data sources** in a **single place**.

Demo

Get in touch

hello@systemticks.de

www.systemticks.de

systemticks

Understanding
Software Systems