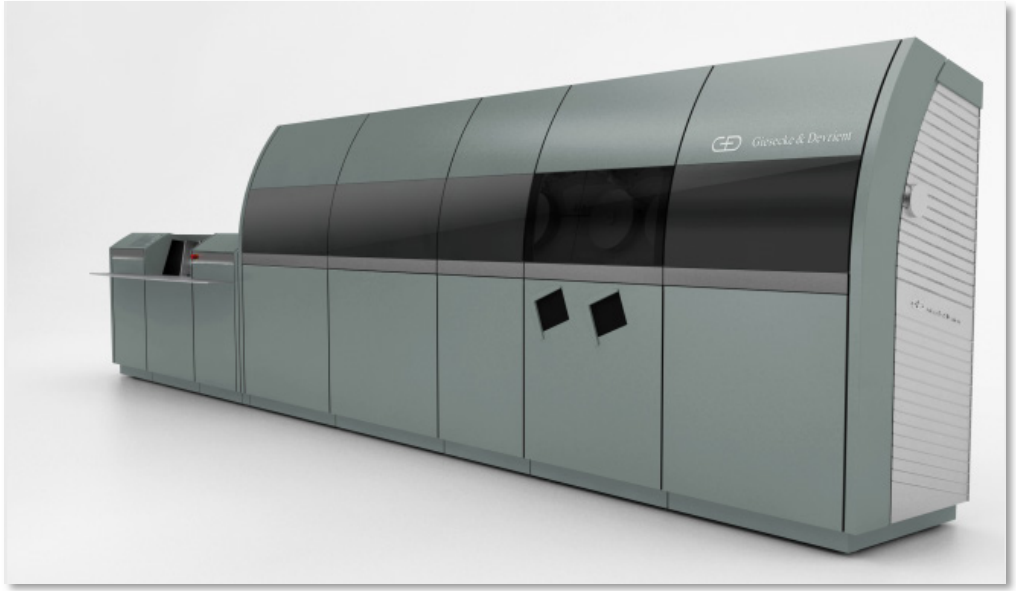
9th UCAAT User Conference on Advanced Automated Testing

Automatic log analysis: Expert knowledge for everyone!

Stephan Schulz , Graham Rawlings

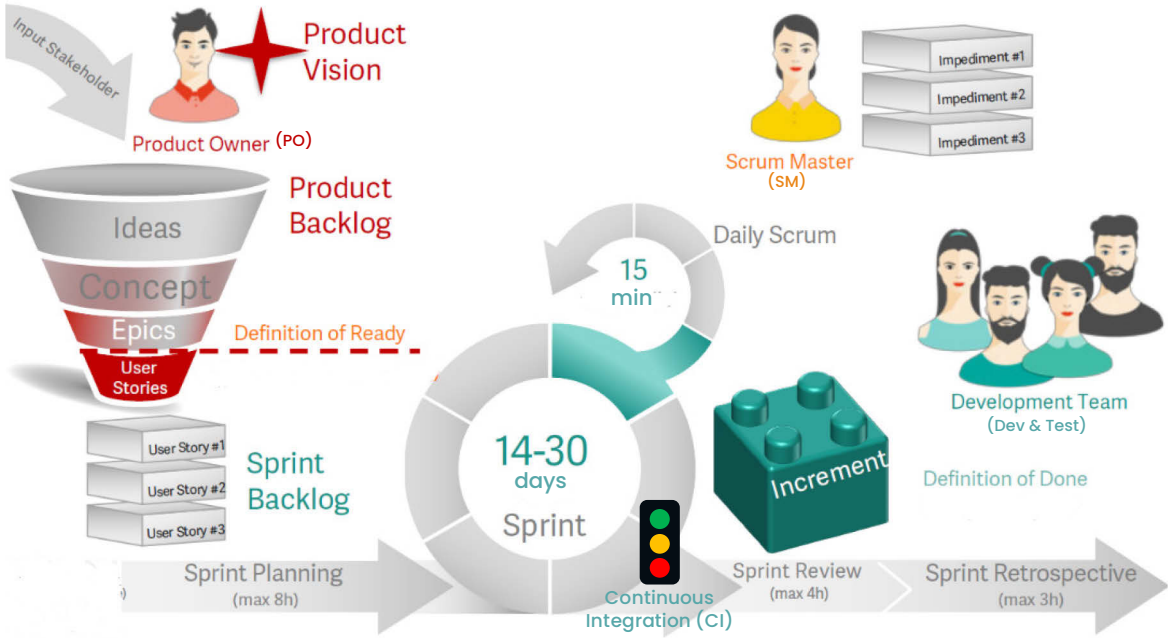
13.9.2022

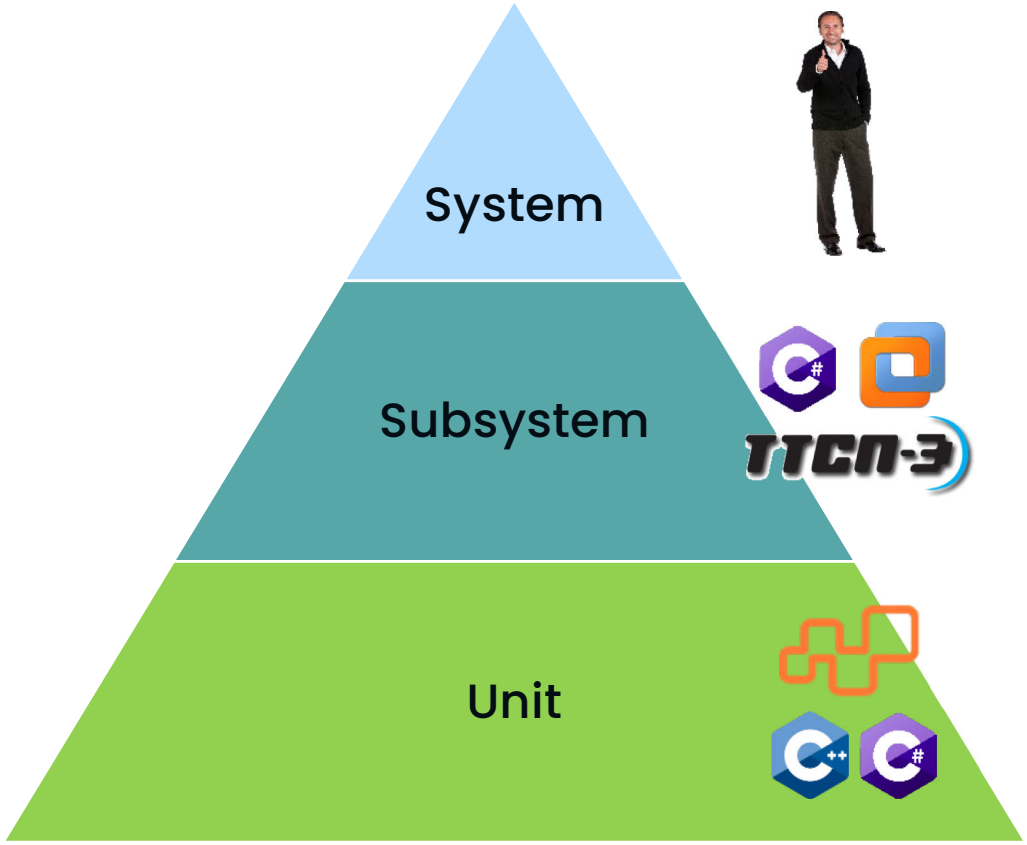
What we test at G+D Currency Technology



- Multiple products based on common product platform that are deployed worldwide in banknote printing, central banks, cash-in-transit centers, casinos, etc
- Each processes millions of banknotes/day 24/7 & is configurable for any currency
- Complex mixed HW/SW system including real-time SW, image processing, embedded control SW, highly concurrent processes, databases, etc.

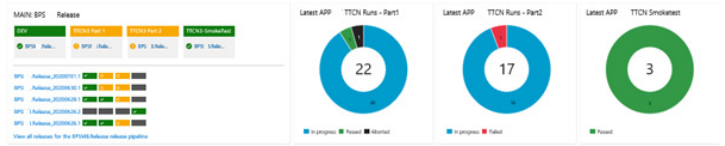
How we develop & test

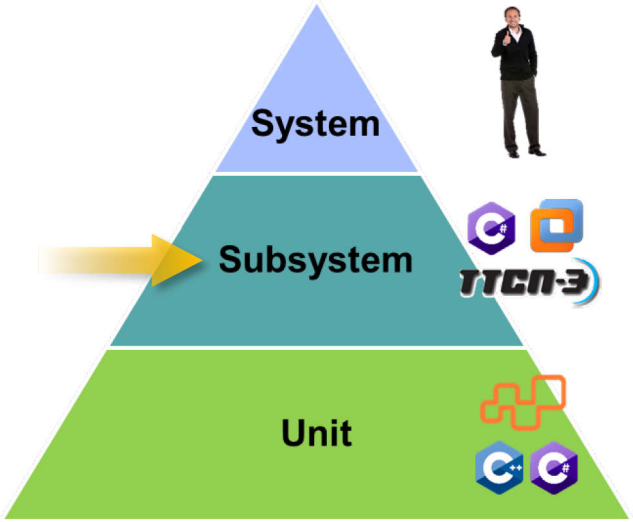




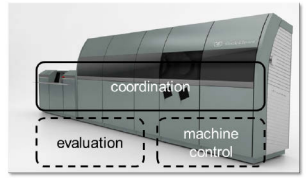
Where we came from

- One automation tester per product to analyze all nightly results for a subsystem
- Our DevOps tool (only) providing a high-level overview of nightly test results
- Automatic tests … but for every test failure *manual* textual log & screenshot *analysis* necessary
- Analysis *results* should be ready every day *in time for daily* at 9 am
  - The more failed tests, the more time required .. the greater the pressure to have these results in time!

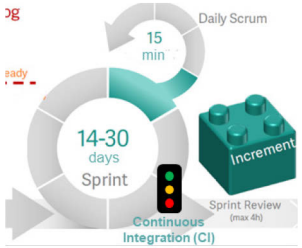


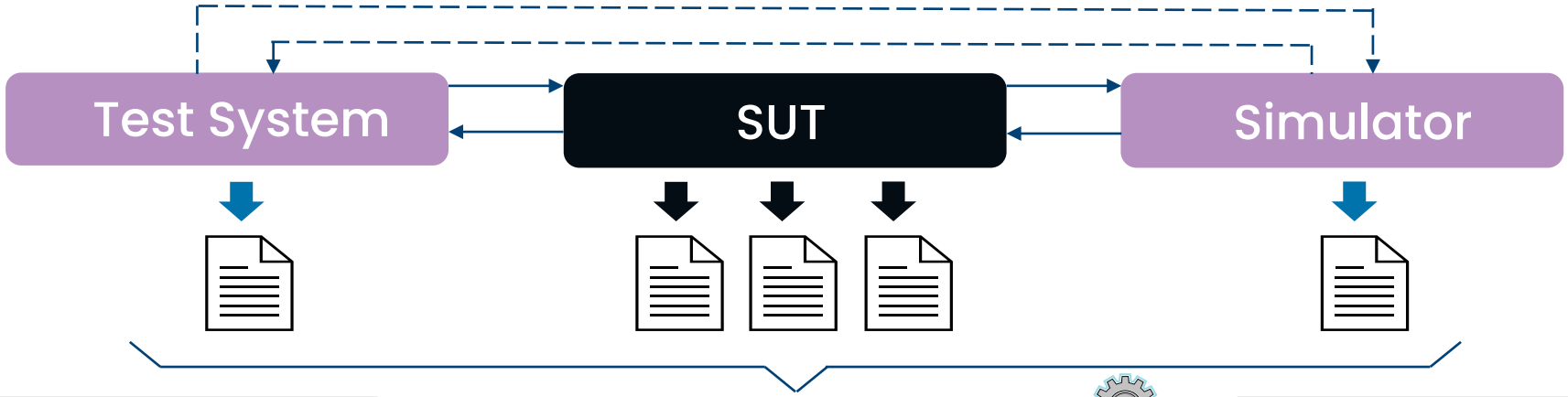


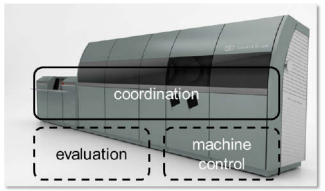
→ This feeling: “Didn’t I [just] see this [pattern] before?!”

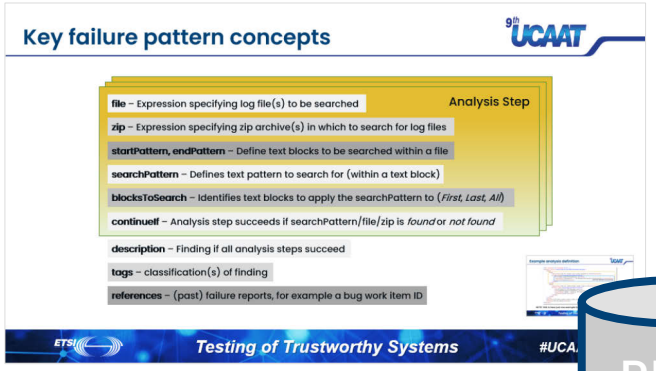


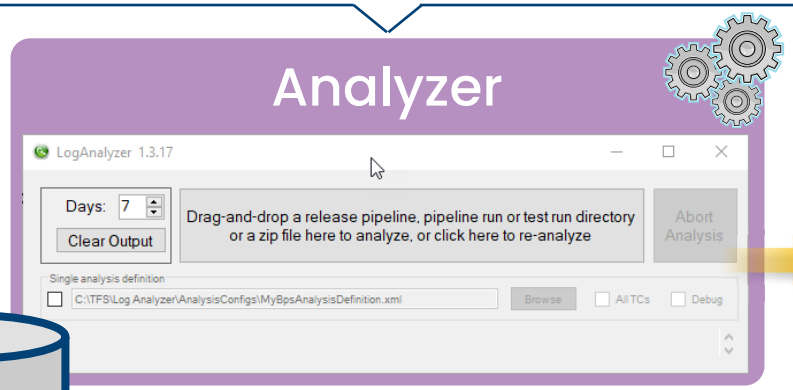
What we did: *Automate* our log analysis!

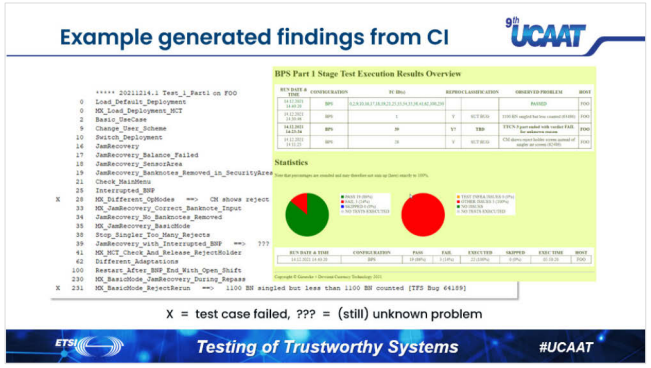








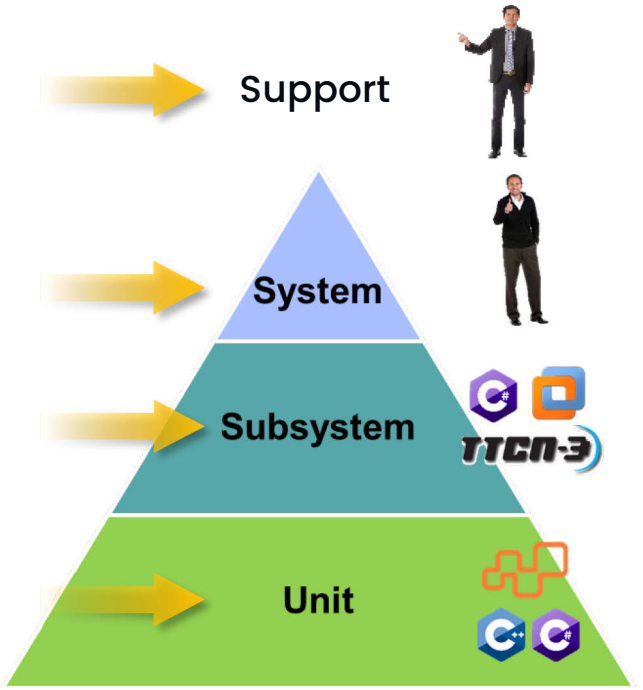




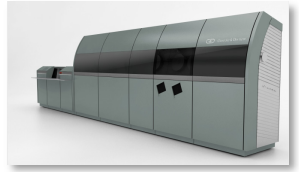
[Link to a more detailed presentation]

HOWEVER …

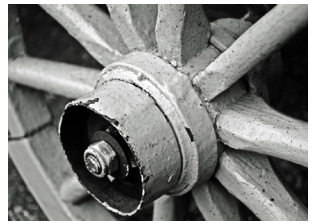
- Fellow automation testers working with other products (same product line) *also analyze* "the same logs"
- MANUAL system TEST, engineering acceptance test, customer acceptance test, etc … *also analyze* "the same logs" to identify issues!
- FIELD ENGINEERS at customer sites *also analyze* "the same logs" to identify issues!
- Even SW DEV *also analyzes* "the same logs" supplied by the support helpdesk to identify issues!



→ "You never analyze alone"!



So what would be our potential benefits?



- Build up knowledge once and reuse it everywhere!
  - Save multiple testers & developers etc from building & applying *the same* analysis definition *in their head* & needing to remember it!
- System test regularly calls a subsystem expert to physically come to the lab to help them understand why the subsystem is not co-operating
  - What is his first question when he arrives?
  - Capture the subsystem expert's knowledge & save him a couple of trips!
  - Imagine this situation during Corona ..

Can you show me the logs?

- Benefits start already way before concrete problems are identified
  - DEV often gets incomplete information from customers through support, leading to multiple iterations before actual log analysis can start
  - Not everyone needs to know how to write analysis definitions but everyone can click a button to get automatic findings, so get *the helpdesk* to identify automatically missing log information!

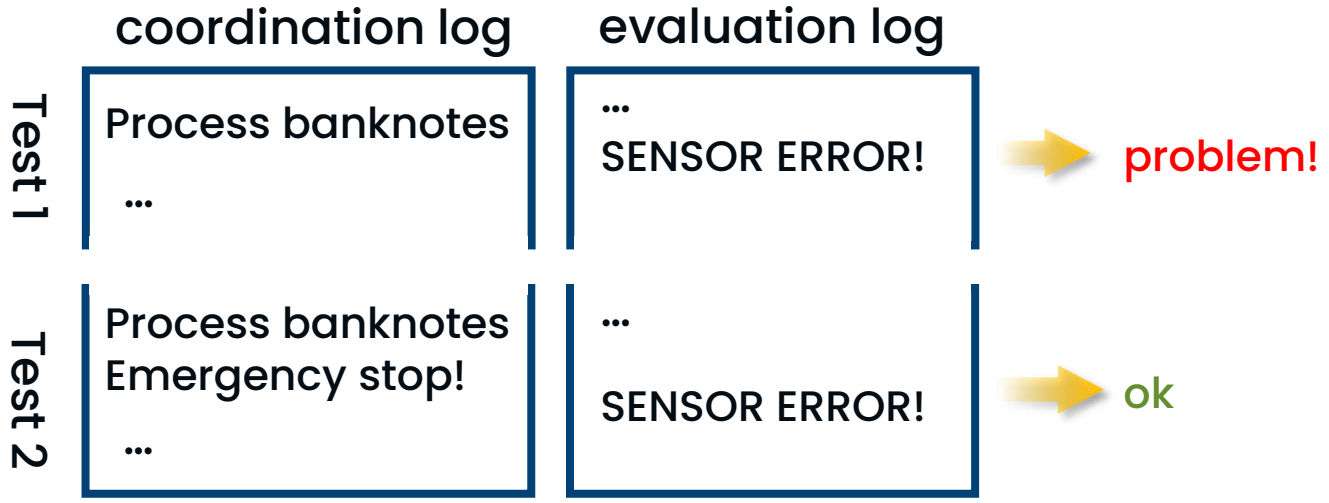
Well – but is it actually so straight forward?

	Automatic Test	Manual Test	Logs from Support
SUT configuration	Fixed	Usually fixed	Not reliably known
Details of SUT interaction	Documented	"Known" by tester	Not reliably known
Verdicts	Intermediate & final	Only final	-
Logs content	Exactly one scenario	One or more scenarios	Production use
Ability to rerun scenario	Generally	Often	Not likely
Need to re-analyze scenario	Every day	Every release	Once
Difficulty to automate analysis	Low	Medium	Medium/High

System level analysis challenges

My analysis definition

```
if SENSOR ERROR and
  not Emergency stop!
then
  Banknote evaluation problem!
```



- Already simple examples with multi scenario logs show that risk of getting false negatives rises sharply
  - With our automatic tests we have so far not observed any false negatives (= patterns not catching problems in logs) – even though at least in theory there is a risk
- In general: a “pollution” of findings for the scenario of interest by other scenarios in the logs!
- Alignment of parallel SUT component logs is not trivial since time settings often differ
  - In general: Understanding operation across (parallel) SUT components is challenging!

Idea: Steer analysis via log visualization

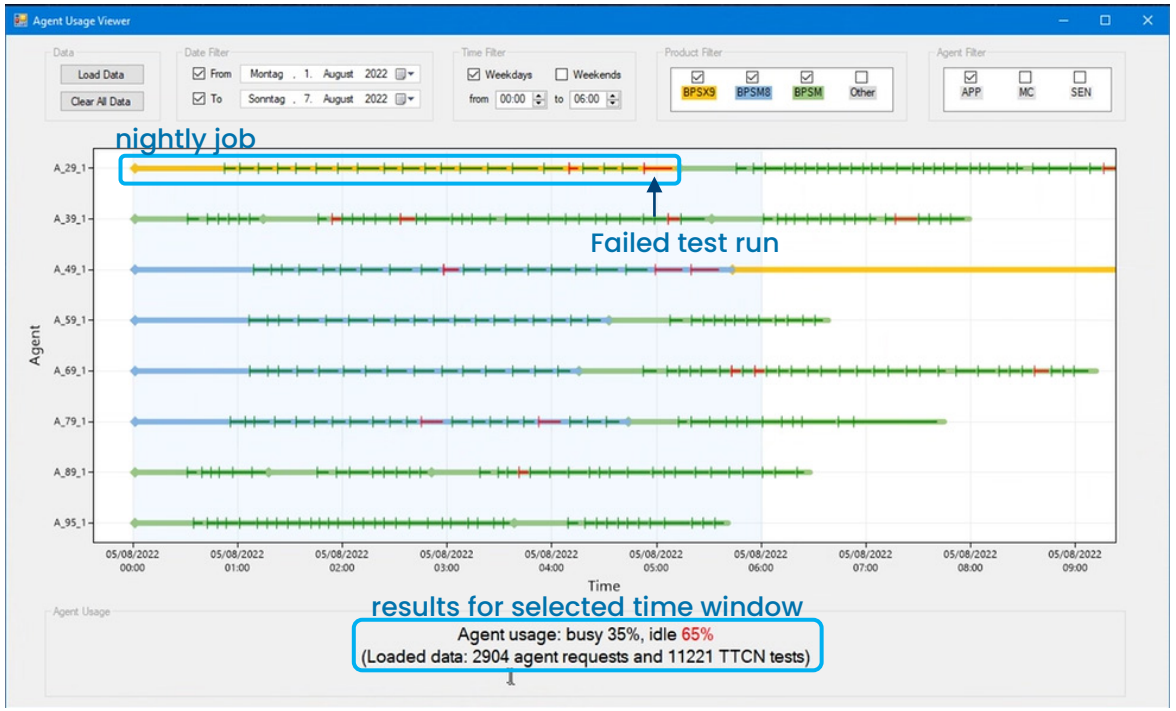
- Visualize key events of SUT component logs on normalized time scale and then run analysis only on a selected window of interest
- Reuse ideas & concepts from recent CI server usage visualization proof of concept

component vs event filters

graphical overlay of event info

event details on hover

graphical selection of time window for further processing



scrollable time scale

zoom in/out

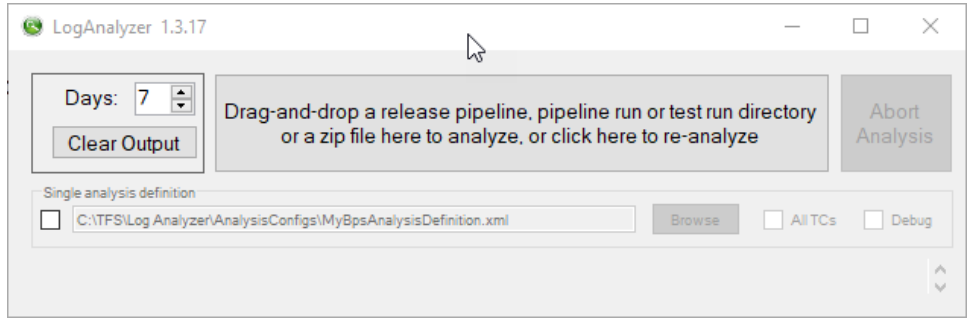
visualization of parallel operation

Conclusions

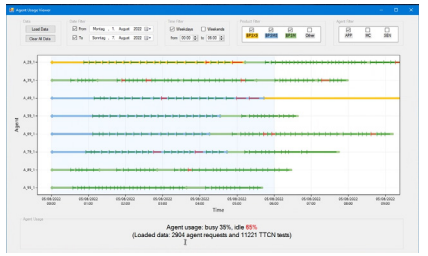
- Automatic log analysis from subsystem testing can also be reused for logs from manual system test or even customers … but a bit more user support is required!
- Analysis of automatic test results turned out to be much easier since logs are generated repeatedly for fixed scenarios within a highly controlled environment
  - In logs produced from manual test or operation the first interest is “what was (really) done?” and “where is my time window of interest which I should analyze?”
- Applying log analysis at system level showed clearly an increase in analysis complexity due to more logs, more log dependencies and more parallelism
  - A visualization of major events across all log data over normalized time is needed to be able to work effectively in our system test

About our tools (ALL NOT COMMERCIAL!)

- G+D Log Analyzer - contact us if you are eager to do this yourself!
  - Compile time 1 second, EXE size 140 KB, requires only .NET Runtime 4.5
  - First version implemented by a tester with DEV background in just two sprints
  - Hints for performance optimizations
    - By default analyze only failed tests
    - Open and parse each file only once, i.e., apply all relevant analysis steps
    - Parse files in zip archives „in place“ without extracting them into the file system
    - Stop executing an analysis as soon as one of its steps fails



- G+D Agent Usage Viewer – contact us if you are eager to do this yourself!
  - First version implemented with similar effort
  - Based on the amazing open source .NET library https://scottplot.net



Any further questions?

Stephan.Schulz@gi-de.com
Graham.Rawlings@gi-de.com

Key failure pattern concepts

Analysis Step

file – Expression specifying log file(s) to be searched

zip – Expression specifying zip archive(s) in which to search for log files

startPattern, endPattern – Define text blocks to be searched within a file

searchPattern – Defines text pattern to search for (within a text block)

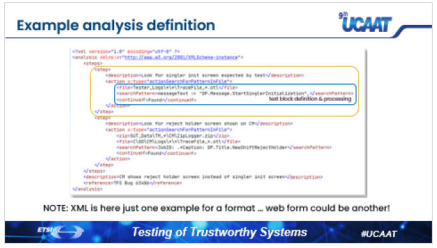
blocksToSearch – Identifies text blocks to apply the searchPattern to (*First*, *Last*, *All*)

continueIf – Analysis step succeeds if searchPattern/file/zip is *found* or *not found*

description – Finding if all analysis steps succeed

tags – classification(s) of finding

references – (past) failure reports, for example a bug work item ID



Example analysis definition

```
<?xml version="1.0" encoding="utf-8" ?>
<analysis xmlns:x="http://www.w3.org/2001/XMLSchema-instance">
    <steps>
        <step>
            <description>Look for singler init screen expected by test</description>
            <action x:type="actionSearchForPatternInFile">
                <file>Tester_Logs\*\*\TraceFile_*.otl</file>
                <searchPattern>messageText := "DP.Message.StartSinglerInitialization",</searchPattern>
                <continueIf>Found</continueIf>
            </action>
        </step>
        <step>
            <description>Look for reject holder screen shown on CM</description>
            <action x:type="actionSearchForPatternInFile">
                <zip>SUT_Data\TM_*\CM\ZipLogger.zip</zip>
                <file>C\GD\CM\Logs\*\*\TraceFile_*.otl</file>
                <searchPattern>JobID: .*Caption: DP.Title.NewShiftRejectHolder</searchPattern>
                <continueIf>Found</continueIf>
            </action>
        </step>
    </steps>
    <description>CM shows reject holder screen instead of singler init screen</description>
    <reference>TFS Bug 63486</reference>
</analysis>
```

text block definition & processing

NOTE: XML is here just one example for a format … web form could be another!

Example generated findings from CI

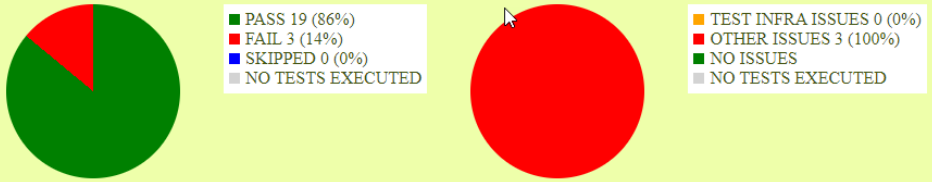
```
      ***** 20211214.1 Test_1_Part1 on FOO
    0 Load_Default_Deployment
    0 MX_Load_Deployment_MCT
    2 Basic_UseCase
    9 Change_User_Scheme
   10 Switch_Deployment
   16 JamRecovery
   17 JamRecovery_Balance_Failed
   18 JamRecovery_SensorArea
   19 JamRecovery_Banknotes_Removed_in_SecurityArea
   21 Check_MainMenu
   25 Interrupted_BNP
X  28 MX_Different_OpModes    ==>   CM shows reject
   33 MX_JamRecovery_Correct_Banknote_Input
   34 JamRecovery_No_Banknotes_Removed
   35 MX_JamRecovery_BasicMode
   38 Stop_Singler_Too_Many_Rejects
   39 JamRecovery_with_Interrupted_BNP   ==>   ???
   41 MX_MCT_Check_And_Release_RejectHolder
   62 Different_Adaptations
  100 Restart_After_BNP_End_With_Open_Shift
  230 MX_BasicMode_JamRecovery_During_Repass
X 231 MX_BasicMode_RejectRerun   ==>   1100 BN singled but less than 1100 BN counted [TFS Bug 64189]
```

BPS Part 1 Stage Test Execution Results Overview

RUN DATE & TIME	CONFIGURATION	TC ID(s)	REPRO	CLASSIFICATION	OBSERVED PROBLEM	HOST
14.12.2021 14:40:20	BPS	0,2,9,10,16,17,18,19,21,25,33,34,35,38,41,62,100,230			PASSED	FOO
14.12.2021 14:30:46	BPS	1	Y	SUT BUG	1100 BN singled but less counted (63486)	FOO
14.12.2021 14:23:34	**BPS**	**39**	**Y?**	**TBD**	**TTCN-3 part ended with verdict FAIL for unknown reason**	**FOO**
14.12.2021 14:11:23	BPS	28	Y	SUT BUG	CM shows reject holder screen instead of singler int screen (62486)	FOO

Statistics

Note that percentages are rounded and may therefore not sum up (here) exactly to 100%.



RUN DATE & TIME	CONFIGURATION	PASS	FAIL	EXECUTED	SKIPPED	EXEC TIME	HOST
14.12.2021 14:40:20	BPS	19 (86%)	3 (14%)	22 (100%)	0 (0%)	05:50:20	FOO

X = test case failed, ??? = (still) unknown problem