Automatic log analysis: Expert knowledge for everyone!

Stephan Schulz, Graham Rawlings
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: \textit{Automate} our log analysis!

Testing of Trustworthy Systems

Analyzer

Example generated findings from CI

Link to a more detailed presentation
• 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 ..

- 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?

<table>
<thead>
<tr>
<th></th>
<th>Automatic Test</th>
<th>Manual Test</th>
<th>Logs from Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUT configuration</td>
<td>Fixed</td>
<td>Usually fixed</td>
<td>Not reliably known</td>
</tr>
<tr>
<td>Details of SUT interaction</td>
<td>Documented</td>
<td>“Known” by tester</td>
<td>Not reliably known</td>
</tr>
<tr>
<td>Verdicts</td>
<td>Intermediate &amp; final</td>
<td>Only final</td>
<td>–</td>
</tr>
<tr>
<td>Logs content</td>
<td>Exactly one scenario</td>
<td>One or more scenarios</td>
<td>Production use</td>
</tr>
<tr>
<td>Ability to rerun scenario</td>
<td>Generally</td>
<td>Often</td>
<td>Not likely</td>
</tr>
<tr>
<td>Need to re–analyze scenario</td>
<td>Every day</td>
<td>Every release</td>
<td>Once</td>
</tr>
<tr>
<td>Difficulty to automate analysis</td>
<td>Low</td>
<td>Medium</td>
<td>Medium/High</td>
</tr>
</tbody>
</table>
System level analysis challenges

My analysis definition

\[
\text{if SENSOR ERROR and not Emergency stop!} \\
\text{then} \\
\text{Banknote evaluation problem!}
\]

- 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
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](https://scottplot.net)
Any further questions?

Stephan.Schulz@gi-de.com
Graham.Rawlings@gi-de.com
# Key failure pattern concepts

<table>
<thead>
<tr>
<th>Analysis Step</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>file</strong> – Expression specifying log file(s) to be searched</td>
</tr>
<tr>
<td><strong>zip</strong> – Expression specifying zip archive(s) in which to search for log files</td>
</tr>
<tr>
<td><strong>startPattern</strong>, <strong>endPattern</strong> – Define text blocks to be searched within a file</td>
</tr>
<tr>
<td><strong>searchPattern</strong> – Defines text pattern to search for (within a text block)</td>
</tr>
<tr>
<td><strong>blocksToSearch</strong> – Identifies text blocks to apply the searchPattern to (First, Last, All)</td>
</tr>
<tr>
<td><strong>continelf</strong> – Analysis step succeeds if searchPattern/file/zip is <em>found</em> or <em>not found</em></td>
</tr>
</tbody>
</table>

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

NOTE: XML is here just one example for a format ... web form could be another!
Example generated findings from CI

BPS Part 1 Stage Test Execution Results Overview

<table>
<thead>
<tr>
<th>RUN DATE &amp; TIME</th>
<th>CONFIGURATION</th>
<th>TC ID(s)</th>
<th>REPRO CLASSIFICATION</th>
<th>OBSERVED PROBLEM</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.12.2021 14:00:20</td>
<td>MPS</td>
<td>0,2,9,10,16,17,18,19,21,25,33,34,35,38,41,62,100,120</td>
<td>PASSED</td>
<td>FO0</td>
<td></td>
</tr>
<tr>
<td>14.12.2021 14:00:46</td>
<td>MPS</td>
<td>1</td>
<td>??</td>
<td>(still) unknown problem</td>
<td>FO0</td>
</tr>
<tr>
<td>14.12.2021 14:29:34</td>
<td>MPS</td>
<td>39</td>
<td>TBD</td>
<td>FO0</td>
<td></td>
</tr>
</tbody>
</table>

Statistics

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

RUN DATE & TIME | CONFIGURATION | PASS | FAIL | EXECUTED | SKIPPED | EXECUTE TIME | HOST |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14.12.2021 14:40:20</td>
<td>MPS</td>
<td>19 (86%)</td>
<td>3 (14%)</td>
<td>22 (100%)</td>
<td>0 (0%)</td>
<td>05:10:20</td>
<td>FO0</td>
</tr>
</tbody>
</table>

Copyright © Grasshopper + Deviant Currency Technology 2021

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