INDUSTRIAL-STRENGTH TEST SYSTEM FOR MULTIPLE DOMAINS
ABOUT ME

› Tool Manager at Ericsson, helping Ericsson sites to develop better software efficiently

› Telecommunication systems
  – Open, standards-based common platform
  – High availability, 99.999 %
  – Broad range of support for both infrastructure and value-added applications
  – Multimedia, network and application processing capabilities

I do not sell processors or tools 😊
Test Automation

- Test cases developed manually
- Frequent re-testing, regression testing

Model Based Testing

- Platform for Test harness
- MBT execution environment

Engine of specialized tools

- Performance testing
- Tariff record precision checking
- Security testing
INGREDIENTS OF SUCCESS

› Partly inherited from the TTCN-3 language

WHAT IS TTCN-3?

› Domain Specific Language for testing
› Abstract language
  - High level, completely platform-independent language
  - Test data and test logic are separated from the environment

› Test cases are re-usable in different development phases and environments
Standard test language
- Continuous language development and maintenance financed by ETSI

Universal language
- Used in several domains: telecom, automotive, avionics, model based testing tool vendors, test equipment vendors, HW vendors, system integrators, standardization/consortia

Designed for automated testing
BENEFITS
Community, Examples, Libraries and more
Why use TTCN-3?

TTCN-3 Test System
Reference Architecture

References

Standardization
Application
Domains
Projects
History

TELECOM
- Industrial use
- Standardization bodies
  - ETSI, 3GPP/GCF, OMA, TETRA
- Test tools manufactures
- Certification program based on TTCN-3
  - WiMAX forum, TETRA (planned)

AUTOMOTIVE
- Car communication systems
- Standardization groups
  - AUTOSAR consortium, MOST cooperation
- Car-To-Car communication

MEDICAL
- Image processing
- HL7 eHealth protocols (Interoperability)

POWER TRANSMISSION AND DISTRIBUTION
- Safe and reliable energy system

FINANCIAL DATAWAREHOUSE
- International bank (functional / regression testing)

AVIONICS
- European Space Agency

RAILWAYS
- Dutch railways
Proven productivity

<table>
<thead>
<tr>
<th>Business parameter</th>
<th>Conventional testing</th>
<th>TTCN-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Productivity</td>
<td>100%</td>
<td>200%</td>
</tr>
<tr>
<td>Impact on Quality</td>
<td>100%</td>
<td>200%</td>
</tr>
<tr>
<td>Impact on CTR</td>
<td>100%</td>
<td>150%</td>
</tr>
<tr>
<td>Reuse</td>
<td>100%</td>
<td>200%</td>
</tr>
<tr>
<td>SUT coverage reached with same effort</td>
<td>60%</td>
<td>90%</td>
</tr>
</tbody>
</table>

## Distribution of project’s time allocated for testing

<table>
<thead>
<tr>
<th>Test activity</th>
<th>Conventional testing</th>
<th>TTCN-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Architecture</td>
<td>7%</td>
<td>8%</td>
</tr>
<tr>
<td>Test Design</td>
<td>10%</td>
<td>7%</td>
</tr>
<tr>
<td>Test case identification</td>
<td>8%</td>
<td>15%</td>
</tr>
<tr>
<td>Test case development</td>
<td>20%</td>
<td>45%</td>
</tr>
<tr>
<td>Cost of quality of test system</td>
<td>7%</td>
<td>7%</td>
</tr>
<tr>
<td>Communication, codecs</td>
<td>8%</td>
<td>8%</td>
</tr>
<tr>
<td>Test environment</td>
<td>25%</td>
<td>0%</td>
</tr>
<tr>
<td>Test Management</td>
<td>7%</td>
<td>5%</td>
</tr>
<tr>
<td>Other</td>
<td>8%</td>
<td>5%</td>
</tr>
</tbody>
</table>

## Software economics

<table>
<thead>
<tr>
<th>Factor</th>
<th>TTCN-3</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill/training</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Coding effort</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Debugging effort</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Results analysis efforts</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Maintenance efforts</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

› **Type** system
  - Rich data type system: any protocol structure can be described
  - Direct import of XSD, ASN.1, IDL and JSON (ongoing)

› **Concept of test(ing) components**
  - Independent behavior simulating a given entity

› **Black box and grey box testing of**
  - message-based and synchronous (API-based) interfaces
  - interfaces with continuous signal and real-time systems

› Handling of non-determinism

› Fine tuning content checking of received messages

› **Fuzzy** testing, built in timer and verdict handling features

› and more …
STRUCTURE OF THE TTCN-3 STANDARDS

Language mappings

- ASN.1
- XSD
- IDL
- JSON (ongoing)

Extensions

- Source code documentation
- Dynamic function references
- Real time, performance
- Advanced parameterization
- Static test configuration
- Continuous signal testing

Core language

- Code structuring
- Types and Test data
- Test configuration
- Test behaviour

Tool implementation

TTCN-3 Runtime Interface

Abstract definition mappings
Java C C++ C#

TTCN-3 Control Interface

Abstract definition mappings
Java C C++ C#
TITAN’S TIMELINE

› 1998: Diploma work on performance testing of IP networks
   Suitable test tool was missing
   -> TTCN-3 compiler started

› Used in several research projects to test or even simulate networks

› 2003: Old test tool needed be replaced in 3G RNC node testing
   -> Titan has been chosen

› 2004: Decision to “productify” Titan

› 2014: Main test tool at many products
   Thousands of users
   Biggest test framework
OPEN SOURCE TITAN

› https://projects.eclipse.org/projects/tools.titan

› 1.6 MLoC in C++, 3 kLoC in Java

› 15 years of development

› Conservative estimate: 100 000+ developer man-hours

› The complete toolset

› A set of protocols:
  – SUT adapters
  – IP-based protocol support
<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Implementation language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titan Designer</td>
<td>Eclipse plugin, TTCN-3 &amp; ASN.1 design (advanced editing, on-the-fly syntax &amp; semantic checking) and build the executable</td>
<td>Java</td>
</tr>
<tr>
<td>Titan Executor</td>
<td>Eclipse plugin, test execution and result reporting</td>
<td>Java</td>
</tr>
<tr>
<td>Titan LogViewer</td>
<td>Eclipse plugin, offline log representation in tabular and graphical (UML SD-like) formats</td>
<td>Java</td>
</tr>
<tr>
<td>Titanium</td>
<td>Eclipse plugin for code quality analysis (identifies code smells, draws module dependency graph)</td>
<td>Java</td>
</tr>
<tr>
<td>TTCN-3 and ASN.1 compiler</td>
<td>Command line parser, semantic analyzer and C++ code generator. Input files can be TTCN-3 and ASN.1.</td>
<td>C++</td>
</tr>
<tr>
<td>xsd2ttcn</td>
<td>Command line tool converting XSD documents to TTCN-3 modules, according to part 9 of the TTCN-3 standard.</td>
<td>C++</td>
</tr>
<tr>
<td>Runtime library</td>
<td>Implementation of TTCN-3 language elements (types, statements, operations, predefined functions etc.) and ETS side of runtime control</td>
<td>C++</td>
</tr>
<tr>
<td>mctr_cli</td>
<td>Command line main controller: runtime control of component distribution and central runtime control of test execution</td>
<td>C++</td>
</tr>
<tr>
<td>makefilegen</td>
<td>Command line Makefile generator</td>
<td>C++</td>
</tr>
<tr>
<td>logmerge</td>
<td>Command line utility to merge the log events events based on their timestamps from the set of textual logfiles produced by the different test components independently.</td>
<td>C++</td>
</tr>
<tr>
<td>logformat</td>
<td>Command line utility to nice-format the textual log files.</td>
<td>C++</td>
</tr>
<tr>
<td>logfilter</td>
<td>Command line utility to post filtering large log files based on the kind of logged events.</td>
<td>C++</td>
</tr>
<tr>
<td>repgen</td>
<td>Command line utility to present not only the formatted log files but the description and TTCN-3 source code of test cases as well as the output of other network monitor programs (like tcpdump) in HTML format.</td>
<td>C++</td>
</tr>
<tr>
<td>tco2lco2</td>
<td>Titan is able to instrument the generated C++ code and output code coverage data in xml during runtime. This command line utility Collects and merges these output files into an LCOV input format.</td>
<td>C++</td>
</tr>
<tr>
<td>Documentation</td>
<td>Installation ~, user ~, programmer reference guides and API specification.</td>
<td>Microsoft Word</td>
</tr>
</tbody>
</table>
Setting up and maintaining the transport connections, and sending / receiving "real" messages and signals are plugins written in C++.

Titan has a C++ API for adaptors that complete the ATS with the connectivity layer(s) between the test system and the SUT.

TCP, UDP, TELNET, SQL, PIPE, SCTP, HTTP, PCAP, LANL2, SIP, Abstract Socket
Titan, the advent of an industrial-strength test system for multiple domains

- **Project explorer area**
- **Editing areas**
- **Outline of the module**
- **Problems found**
- **On-the-fly and compiler analysers**
- **Console view for the builder**
Fast compilation, incremental re-compilation

Supported APIs:
- SUT adaptor (C/C++) API
- External C/C++ functions: own utilities and codec can be added easily
- Codec generators for binary (bit-oriented) and textual protocols
- Built-in loggers and API to add logger plugins and textual protocols
- Interworking with other languages like Java, Python etc.
Titan, the advent of an industrial-strength test system for multiple domains
TEST EXECUTION

› High performance runtime

› Detailed logging with configurable verbosity by event types

› Use of macros and environment variables for runtime configuration, self-configuring test execution

› Distributed test execution on multiple platforms: Linux, Solaris and Windows
Test case list and verdicts extracted from the log.

Graphical view of test case log.

View is selectable.

Jumping to source code line producing the highlighted log event.

Value view to analyse message content.
TEST RESULT ANALYSIS
(TABULAR VIEW)
CMD LINE SUPPORT

- Command line interface and tools for the complete edit-compile-execute-analysis workflow
- Manual or automated build of executable from sources
- Manual or automated test execution
- Configurable on line monitoring of test execution progress & verdicts
- Post-execution log collection and merging
- Log post-processing utilities (filtering, nice-formatting, convert to html etc.)
INCREASING EFFICIENCY BY TEST AUTOMATION

Source: Ina Schieferdecker, ICTSS 2010
› All-in-one solution for your testing needs

› Supports about 140 protocols (215+ protocol variants)

› Several APIs allow adopting to virtually any environment

› Quick to develop new protocol support and quick adaptation to protocol changes
More than ten years of development, thousands of active users

Secure investment: based on a standard test language

Well suited to both traditional V process and agile development processes

Allows testing at an early phase of development: decreased cost, improved product quality

Faster development of tests with an easy-to-learn intuitive test tool and language

Multi-purpose: functional and non-functional testing (performance, security)

High degree of test re-usability
15 years
50 billion connected devices

25 years
5 billion connected people

100 years
1 billion connected places

Connections (billion)

1875 2000 1975

2020
Using Titan in Product Development Phases

- **Network level**: Used in end-2-end testing scenarios
- **System level**: System integration verification and key usage in load and performance testing
- **Function level**: The #1 most used function testing method in Ericsson
- **Unit level**: Model based testing
- **Unit level**: Used as one out of several unit (basic) testing techniques

Titan, the advent of an industrial-strength test system for multiple domains | Public | © Ericsson AB 2014 | 2014-07-25 | Page 31
Each simulated node is a separate TTCN-3 component, representing a different protocol stack, and communicating with the node or system under test (SUT) over a test port.
• Open Innovation and advanced features
• No Lock-in: you or third parties can add features
• Open source with commercial support
• No license fees
• Industrial user community driven
• World class intellectual property management for open source

• Very long term support
• Systematic maturity assessment
• Interoperability
• Accelerates product development
• Technology platform
• Designed for extensibility and adaptation to your context
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATS</td>
<td>Abstract test suite</td>
</tr>
<tr>
<td>ETS</td>
<td>Executable test suite</td>
</tr>
<tr>
<td>pETS</td>
<td>Parameterized executable test suite</td>
</tr>
<tr>
<td>ASP</td>
<td>Abstract Service Primitive</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunication Standards Institute</td>
</tr>
<tr>
<td>3GPP</td>
<td>3rd Generation Partnership Project</td>
</tr>
<tr>
<td>ITU-T</td>
<td>International Telecommunication Union, Telecommunication Sector</td>
</tr>
<tr>
<td>MTC</td>
<td>Main test component</td>
</tr>
<tr>
<td>PTC</td>
<td>Parallel test component</td>
</tr>
<tr>
<td>IUT</td>
<td>Implementation under test</td>
</tr>
<tr>
<td>SUT</td>
<td>System under test</td>
</tr>
<tr>
<td>TTCN-2</td>
<td>Tree and Tabular Combined Notation version 2</td>
</tr>
<tr>
<td>TTCN-3</td>
<td>Test and Test Control Notation version 3</td>
</tr>
<tr>
<td>MBT</td>
<td>Model-based testing</td>
</tr>
</tbody>
</table>