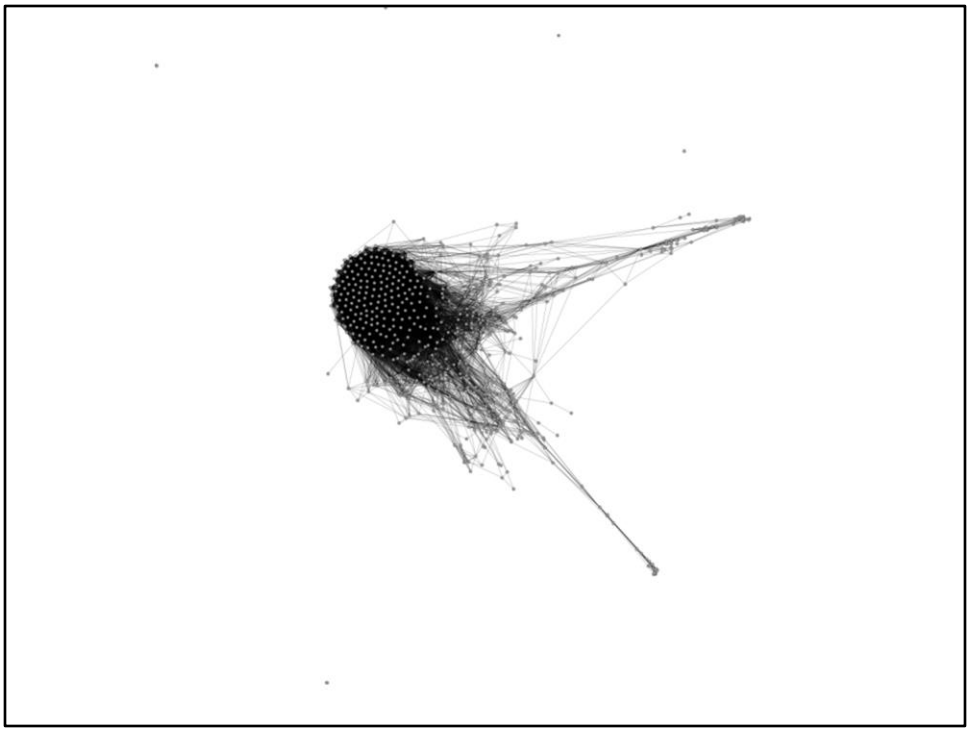
Technical Debt of Standardized Test Software

at 7th International Workshop on Managing Technical Debt



When we first visualized the architecture of a large test system we got this picture.

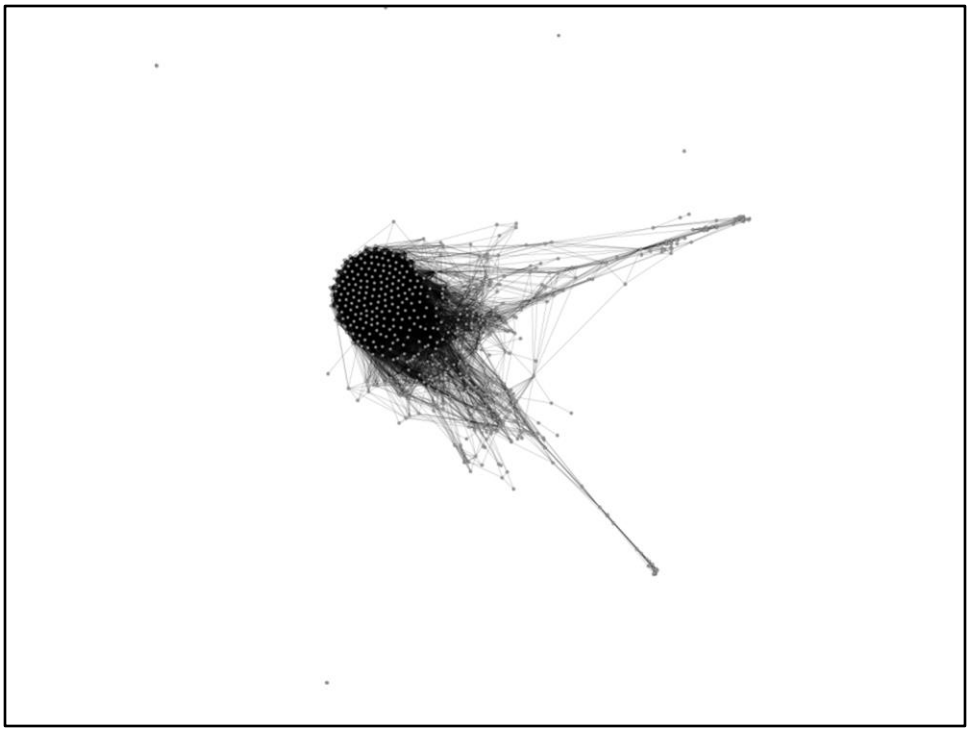
Although it looks large, complex and badly architected …
… we were much more surprised by the lack of research in this field.
We could not find results describing how test systems should be structured.

First things first, let us introduce ourselves.

Our project is part of a cooperation between Ericsson and the Eotvos Lorand University.
Attila Kovacs is a professor at the university.
Kristof Szabados is an Ericsson employee and a Ph.D student at the university.

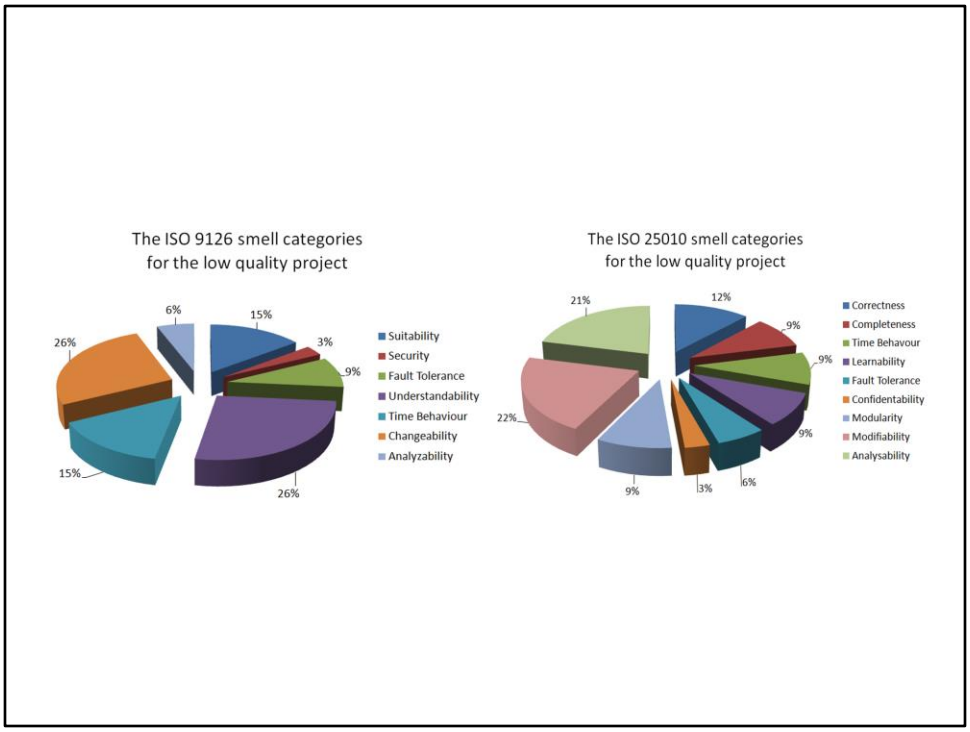
Personal pages of the authors:
http://compalg.inf.elte.hu/~attila/
https://www.researchgate.net/profile/Kristof_Szabados

The home page of the project: http://compalg.inf.elte.hu/~attila/TestingAtScale.html



The network in the slide represents an architecture of a test system. It has small world and scale-free properties.
If we select any 2 nodes, the shortest path between them will contain at most 5 other nodes.
The distribution of the connection follows the power law. Just like the Internet, the human brain and other software systems.

For more information:
K. Szabados, Structural Analysis of Large TTCN-3 Projects in proceedings of: Testing of Software and Communication Systems, 21st IFIP WG 6.1 International Conference, TESTCOM 2009 and 9th International Workshop, FATES 2009, Eindhoven, The Netherlands, November 2-4,
2009.

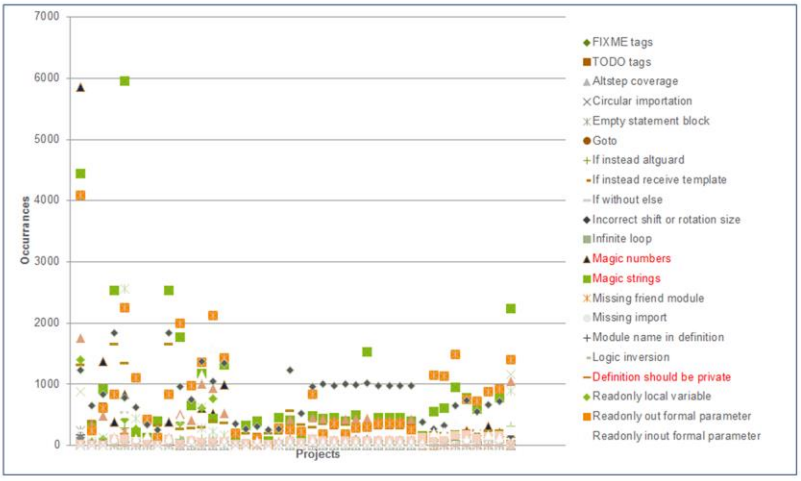


We look at test system as software systems (that happen to be used for testing). We are interested in the properties of test systems as software systems: maintainability, reliability, performance, etc…

For this reason we have translated ISO 9126 and ISO 25010 software requirements for tests written in TTCN-3.
We looked at several tools analyzing software systems: FindBugs, FxCop, PMD, CheckStyle, etc…

We have defined and implemented 40+ code smells, covering all main categories of the ISO 9126 and ISO 25010 software quality standards.

Related publication:
A. Kovács and K. Szabados, Test software quality issues and connections to international standards in Acta Universitatis Sapientiae, Informatica, 5/2013. pages 77-102.



We have analyzed all test systems available on ww.ttcn3.org

We found hundreds, in some cases thousands of code smell instances in standardized test suites.
The most popular being:

- Magic strings and numbers
- Un-initialized local variables
- Unused global definitions
- Definitions that should be private, but are not set so

Related publication:
A. Kovács and K. Szabados, Advanced TTCN-3 Test Suite validation with Titan, 2014, In Proceedings of the 9th International Conference on Applied Informatics, Vol. 2, pages 273-281.

Smell	Easy	Average	Hard
goto	1	5.5	26
circular importation	2	12	80
missing imported module	0	0.5	3.5
unused module importation	0	0.5	1
non-private private definitions	0	0.5	4.5
visbility in name	0	0.5	4.5
unnecessary negation	0	0.5	3.5
module name in definition	0	1	3.5
type in definition name	0	1	2
magic constants	0	0.5	3
infinite loops	0	1	3.5
uninitializaed variable	0	0.5	2
size check in loop	0	1	5
consecutive assignments	0	1	6
read-only variables	0	2	5

To understand the cost of issues in the source code of TTCN-3, we asked experts in the field (test system architects, test developers and test maintainers) to help us estimate the cost of fixing an instance of an issues.
Using the Delphi method we have estimated 3 Man-Hour values for all measured code smell types.

- Easy: Fixing an instance of the code smell if it is the easiest situation they can think of. For example removing a dependency that is not actually used to import definitions.
- Average: Fixing an instance of the code smell, that is common in real life, according to their experiences.
- Hard: Fixing an instance of the code smell if it is the most complex situation they can think of. For example removing a dependency, requiring architectural restructuring.

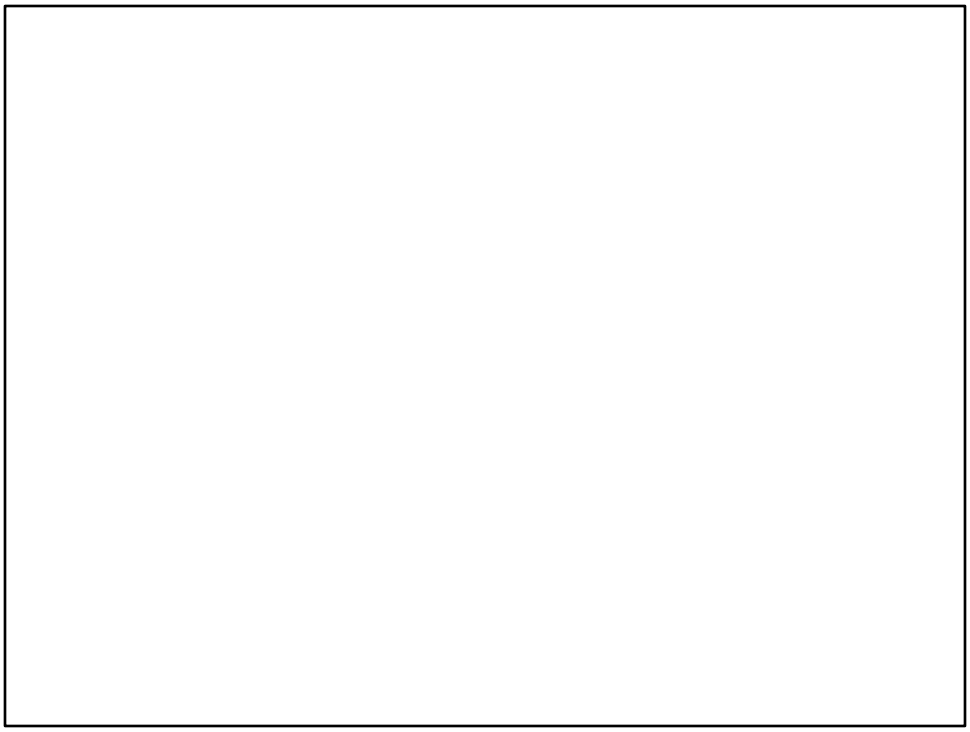
Project No.	Identifier	*Min*	*Avg*	*Max*
1	36.523-3v10.3.0	1528	20659.5	91282.5
2	34.229-3v9.7.0 / IMS34229	392	4053.5	16886
	34.229-3v9.7.0 / IMS36523	580.5	6767	30392.5
3	TS 102 624-3	1699	13262	63426.5
4	TS 102 545-3	2552	14979.5	69307
5	TR 103 200	163	1928.5	8949.5
6	TS 102 027-3	1335	7126	39363
7	TS 101 580-3*	833.5	7438	33715
	TS 101 606-3*	307.5	2979.5	13382.5
	TS 102 790-3*	729.5	6529	28956.5
	TS 102 891-2*	705.5	6237.5	28136
	TS 186 001-2	844	9179	40899
	TS 186 001-4*	557	5459	24966.5
	TS 186 002-4	1326.5	12378	52104.5

The slide shows a part of our measurements for all publicly available test systems from www.ttcn-3.org.
It can be seen that even if all code smell instances are the easiest to correct, the total cost of correction would be month in most of the test systems.
If they would the hardest to solve, the total technical debt would be years to solve.

11	TS 102 859-3*	193	2082.5	9175
	TS 102 868-3 ver 1.1.1*	186	1652	7615.5
	TS 102 869-3 ver 1.2.1*	187	2093.5	10218
	TS 102 870-3 ver 1.1.1*	137	1350.5	6158
	TS 102 871-3 ver 1.1.1*	161.5	1927.5	8796.5

It is important to research the quality of these test suites, as their quality might have a real-file effect.

Project no. 11 is the “INTELLIGENT TRANSPORT SYSTEMS”, being in development right now.



We believe this to be a field not yet fully researched, an interesting opportunity for further study.

Our tools is called Titanium, and is available in open source, as part of the Titan toolset: https://projects.eclipse.org/projects/tools.titan/downloads