



LSB Infrastructure: Testing and Analytics

Vladimir Rubanov

Project Manager

Institute for System Programming of RAS

Linux Verification Center (linuxtesting.org)



About the LSB Infrastructure Program

- Started in September 2006
- Contractor: Institute for System Programming of the Russian Academy of Sciences (ISP RAS), Moscow, Russia
- People from the Linux Verification Center, which is a part of ISP RAS





Program Areas

- **Infrastructure SW Systems**

- Main LSB Database & Scripts
- LSB Navigator
- LSB Certification System
- Test Execution Frameworks (ATK / DTK Managers)

- **New Tests for LSB Interfaces**

- 3 grades of quality – shallow, normal, deep.
- 3 Test Development Frameworks

- **Investigation and Analytical Tasks**

● ● ● | Current Test Coverage (1)

○ **Optimistic** (interface is covered if it is called at least once):

● Total -	15.9%
● QT (3 & 4) -	3.5%
● Sub Total (no qt) -	38.3%
• Core -	33.3%
• C++ -	20.7%
• Desktop (no qt) -	44.7%

● ● ● | Current Test Coverage (2)

○ **More adequate** (interface is covered if it has a special test for it):

- **Total - 9.2%**
 - **QT (3 & 4) – 0.8%**
 - **Sub Total (no qt) - 24.6%**
 - Core – 27.7%
 - C++ – 5.9%
 - Desktop (no qt) – 29.7%



Testing Quality Grades

- **Shallow** – trivial tests with the only guaranteed purpose of ensuring the interface does not crash being called with some particular correct parameters and in the correct environment.
- **Normal** – most reasonable level of testing achievable by tests written in plain C. Tests check main functionality well.
- **Deep** – this is the level when most of the specification assertions are tested in various conditions/states. This is usually for critical software.



Test Development Frameworks

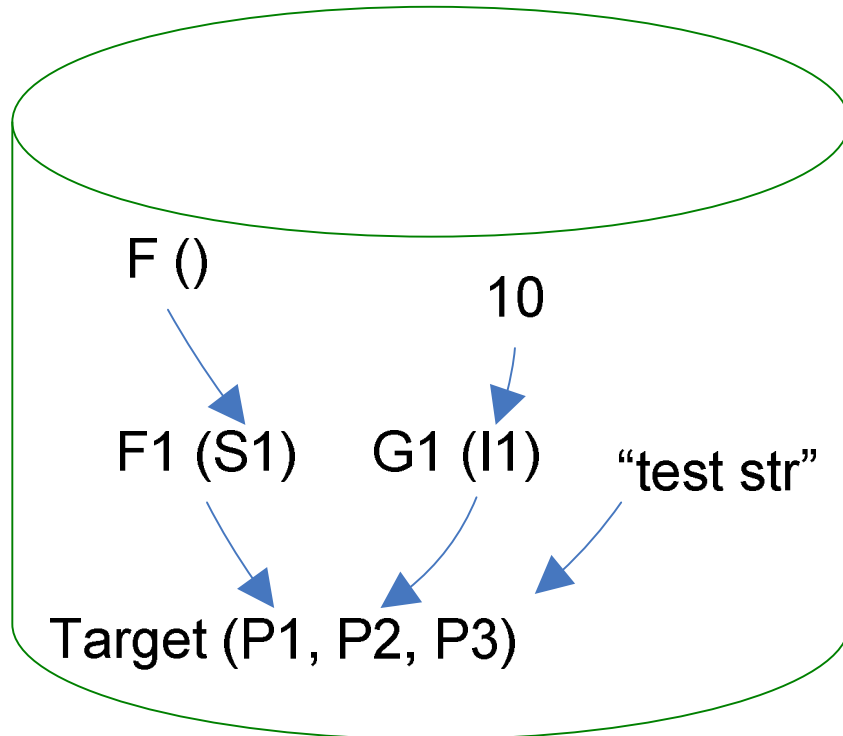
- **3 testing quality grades = 3 test development frameworks.**
- **Shallow Testing**
 - **AZOV Technology & Tools:** tests are automatically produced from LSB DB being cleaned up and extended with additional information specific for tests generation.
- **Normal Testing**
 - **T2C Technology & Tools:** almost plain C with some templates, macros and library calls.
- **Deep Testing**
 - **UniTESK Technology and Tools:** model based testing with formal specifications.



● ● ● | *Shallow* Testing Framework

- **AZOV Technology and Tools**
- Based on the extended LSB Database:
 - New fields in existing tables.
 - 7 new tables specific for test data.
- A web interface for editing, analyzing and managing the info.
- Test generator creates shallow tests automatically from the extended DB information.

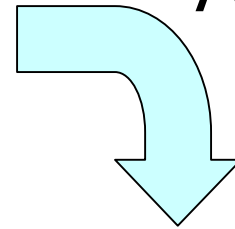
● ● ● | *Shallow Testing Example*



Web-ui

Developer

AZOV Generator



```
S1 = F ()  
P1 = F1 (S1)  
I1 = 10  
P2 = G1 (I1)  
P3 = "test string"  
Res = Target (P1, P2, P3)  
CHECK (Res != error)
```



Normal Testing Framework

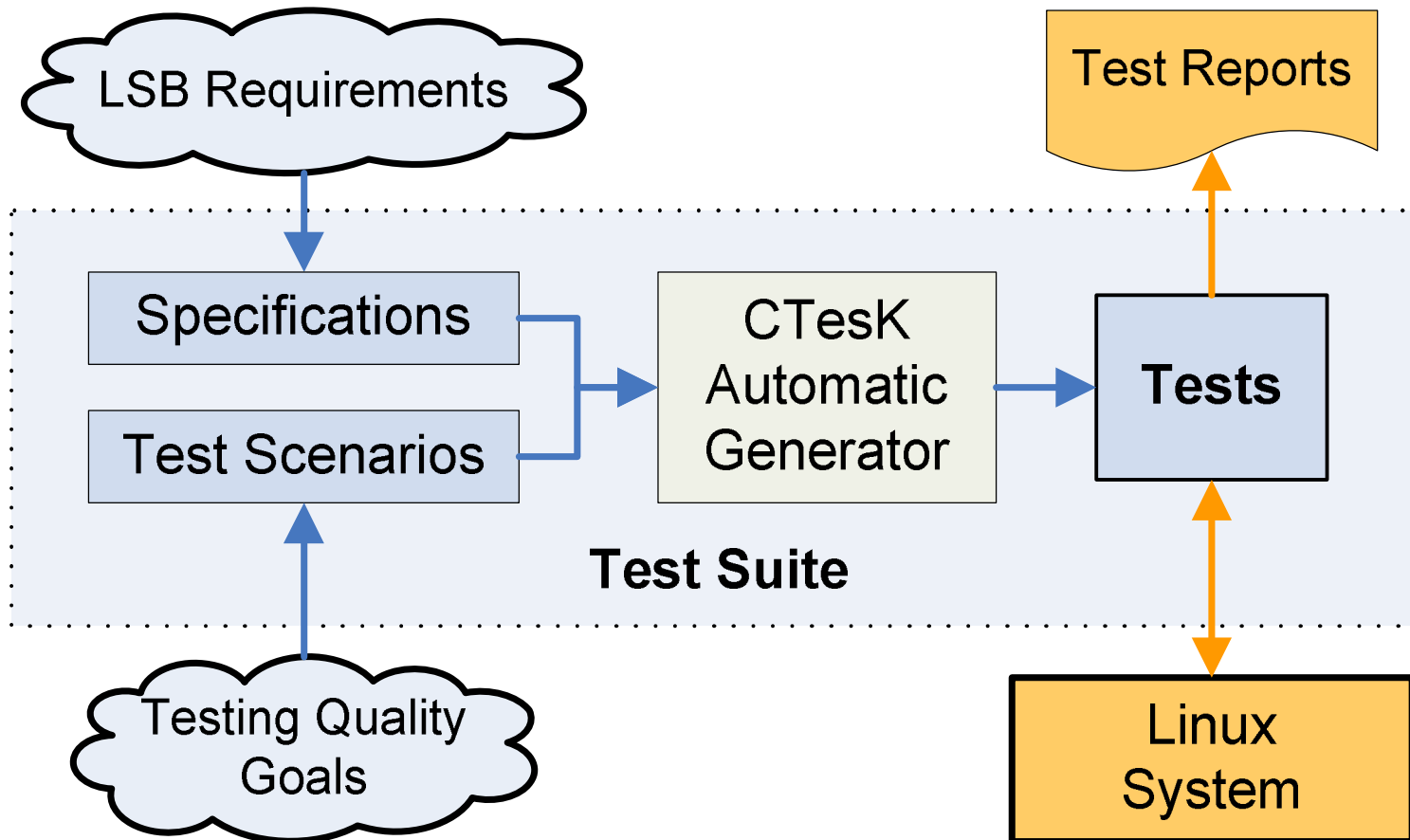
- **T2C Technology and Tools**
- Templates are in T2C format, which is basically C with extensions and additional macros and library calls available.
- .t2c files are automatically translated into two variants of each test case:
 - To be run under TET and DTK Manager
 - Independent pure C test case for debugging



● ● ● | *Deep* Testing Framework

- **UniTESK Technology and Tools**
- Model based testing.
- Formal specifications and test scenarios in SeC language.
- Pre and post conditions.

Deep Testing Framework



● ● ● | Linkage to Assertions

- Specification's text is divided into atomic assertions (requirements).
- **Assertions catalog** is created:
 - {ID, except, rephrased text}
- Each check in the test code is linked to one or many assertions.
- If such check fails – user gets a message with the text of assertion violated as well as additional trace about particular mismatch like “XX returned, while YY expected”.

● ● ● | Missing Spec Problem

- Some LSB interfaces do not have documentation...
- Documentation is a must for normal testing to extract assertions (requirements) to check.
- Three ways here:
 - Ask upstream to write documentation.
 - Understand what the interface does and write documentation.
 - Forget it (no normal test for the interface).



Test Development In Progress

- **Shallow** testing in progress
 - a pilot for Qt3.
- **Normal** testing in progress:
 - GTK_glib (832 total interfaces)
 - GTK_gmodule (8 total interfaces)
 - GTK_atk (222 total interfaces)
 - GTK_gobject (314 total interfaces)
 - fontconfig (160 total interfaces)
- **Deep** testing:
 - OLVER (for LSB Core) adaptation and development.



LSB 3.2 (September) Plans

- o New **'normal'** tests for:
 - GTK gmodule
 - GTK glib
 - GTK atk
 - fontconfig

● ● ● | Test Development Strategy

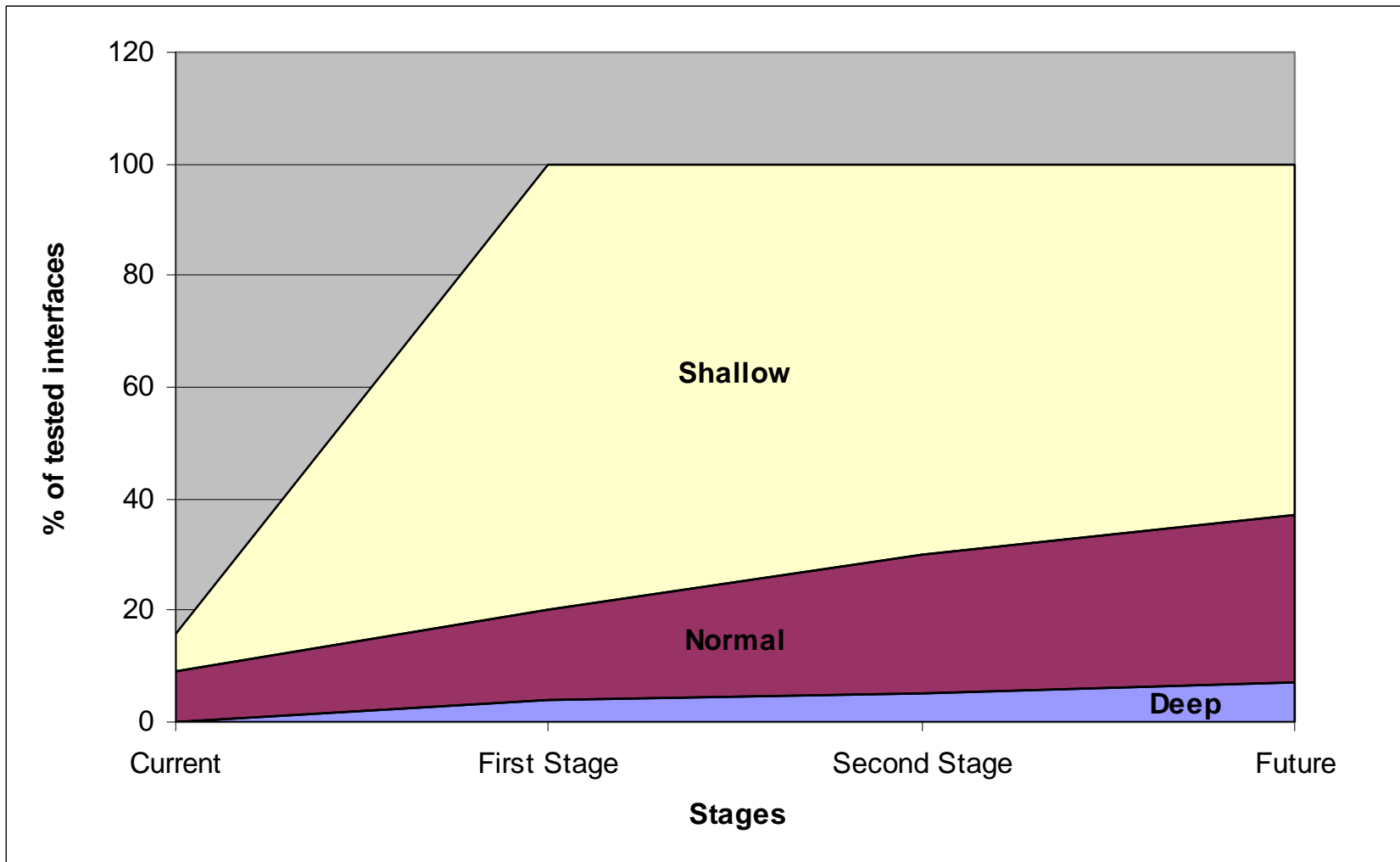
○ **First Stage (LSB 4.0) –** eliminate the gaps in interfaces without tests at all:

- Shallow – 80% of interfaces (50% of libs).
- Normal – 16% of interfaces (30% of libs)
– GTK & Graphics focus.
- Deep – 4% of interfaces (20% of libs)
– most of the Core.

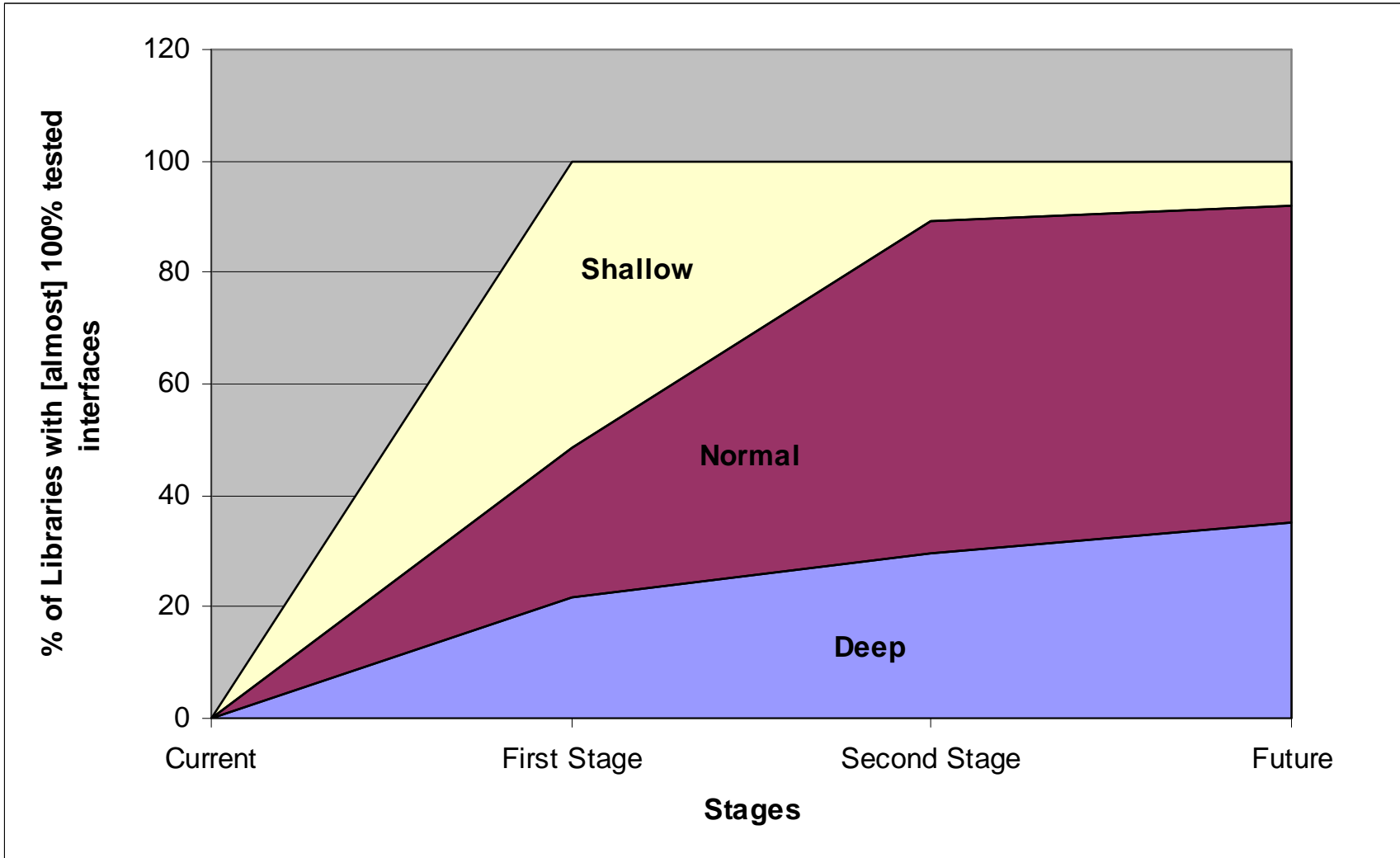
○ **Second Stage (mid 2009) –** cover most of the LSB libraries (almost entire C part):

- Shallow – 70% of interfaces (10% of libs)
– mainly Qt & C++
- Normal – 25% of interfaces (60% of libs)
– Desktop excl. Qt
- Deep – 5% of interfaces (30% of libs) - All Core₇

● ● ● | Test Strategy: Interfaces



Test Strategy: Libraries



● ● ● | Test Development Futures

- Maintain and improve the test coverage:
 - Upgrade shallow tests into normal tests for few remaining important libraries.
 - Cover individually selected most important interfaces by deep tests.
 - Ensure no testing gaps occur in newly added interfaces.
 - Keep the tests up-to-date to upstream movement.
 - Maintain the tests and improve reporting messages.

● ● ● | Analytical Work (1)

- Dig for new knowledge:
 - Analysis of Tests <-> Interfaces mapping.
 - Analysis of Documentation <-> Interfaces mapping.
- Find and fix various issues in LSB ecosystem:
 - DB data clean up and augmentation.
 - Discover problems in existing documentation.
 - Analyze problems in implementations found by tests.
 - Analytical activities triggered by issues raised in LSB Bugzilla and mailing lists.



Analytical Work (2)

- Community relations:

- Communicate with upstream developers to discuss and fix problems in documentation and implementations.
- Work with ISVs to get info about their applications uploaded to the LSB Navigator.
- Work with distribution vendors to analyze their distributions and get the info uploaded to the LSB Navigator.



Analytical Work (3)

- LSB Analysis:

- Identify interfaces that are in the standard by mistake.
- Identify interfaces that should be added to the standard in the already included libraries.



Analytical Work Example (1)

- Note: only 250 applications analyzed.
- Only **4** small libraries in LSB with 100% of interfaces in use – libcrypt, libdl, libpam, libutil.
- **Top 5** big (>100 interfaces) libraries by used interfaces fraction:
 - libGL - 89% interfaces in use
 - libc - 79% interfaces in use
 - libgtk-x11 - 75% interfaces in use
 - libgobject - 71% interfaces in use
 - libglib - 70% interfaces in use
- **All** LSB libraries and all LibGroups have at least 1 interface in use.

● ● ● | Analytical Work Example (2)

- Note: **250** applications analyzed.
- **62%** of LSB interfaces are not called by any of these apps.
- Most used interfaces:
 - strlen - 235 apps
 - strcmp - 231 apps
 - free - 225 apps
 - memcpy - 225 apps
 - fclose - 222 apps
 - exit - 221 apps
 - malloc - 221 apps



LSB Big Clean-Up

- **Technical clean-up**
 - LSB DB data consistency and completeness.
- **Documentation clean-up**
 - Ensure all interfaces have good documentation.
- **Testing clean-up**
 - Ensure all interfaces have proper tests.
- **Interfaces clean-up**
 - Ensure all interfaces in the standard are relevant and serve the purpose of the standard.



More Info & Contacts

- **Mailing list:**

- lsb-infrastructure@lists.freestandards.org

- **Project Wiki:**

- <http://ispras.linux-foundation.org>

- **Vladimir Rubanov**

- vrub@ispras.ru