8th Central and Eastern European Software Engineering Conference in Russia - CEE-SECR 2012

November 1 - 2, Moscow

Testing of Changes in Software System Based on Source Code Coverage

Alexey Salmin

Alexander Stasenko

Regression testing

- Testing process which is applied after a program is modified
- Testing results are compared to the previous ones (reference results)
- The main purpose is to find new bugs (regressions)
- Bug-fixes are appreciated

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice
intel

Regression testing

"As a consequence of the introduction of new bugs, program maintenance requires far more system testing per statement written than any other programming. ***Theoretically, after each fix one must run the entire bank of test cases previously run against the system, to ensure that it has not been damaged in an obscure way.*** *In practice such regression testing must indeed approximate this theoretical ideal, and it is very costly."*

[Frederick P. Brooks, Jr. ***The Mythical Man-Month]***

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

Pre-commit testing

- It’s better to keep serious bugs out of trunk and important branches
- The best way is to require for every change to pass some essential set of tests *before* the commit
- Good testing infrastructure allows developer to type e.g. “test_my_changes.sh” in the workspace and get the list of “new passes” and “new fails” on all platforms
- **Too long pre-commit testing significantly slows down the development process**
- **Too small pre-commit testing leads to unstable codebase**

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

intel

How to form a pre-commit testing

- Tests for important features
- “Importance” is questionable
- Tests are not always bound to specific features
- Tests that had failed before
- Will never catch a truly “new fail”
- Test failed once will stay there forever
- Redundant: single bug often causes a number of fails
- Unique minimal defect reproducers
- More efficient but still one step behind
- Minimal testing for each component (or even feature)
- Reliable but may be too big

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

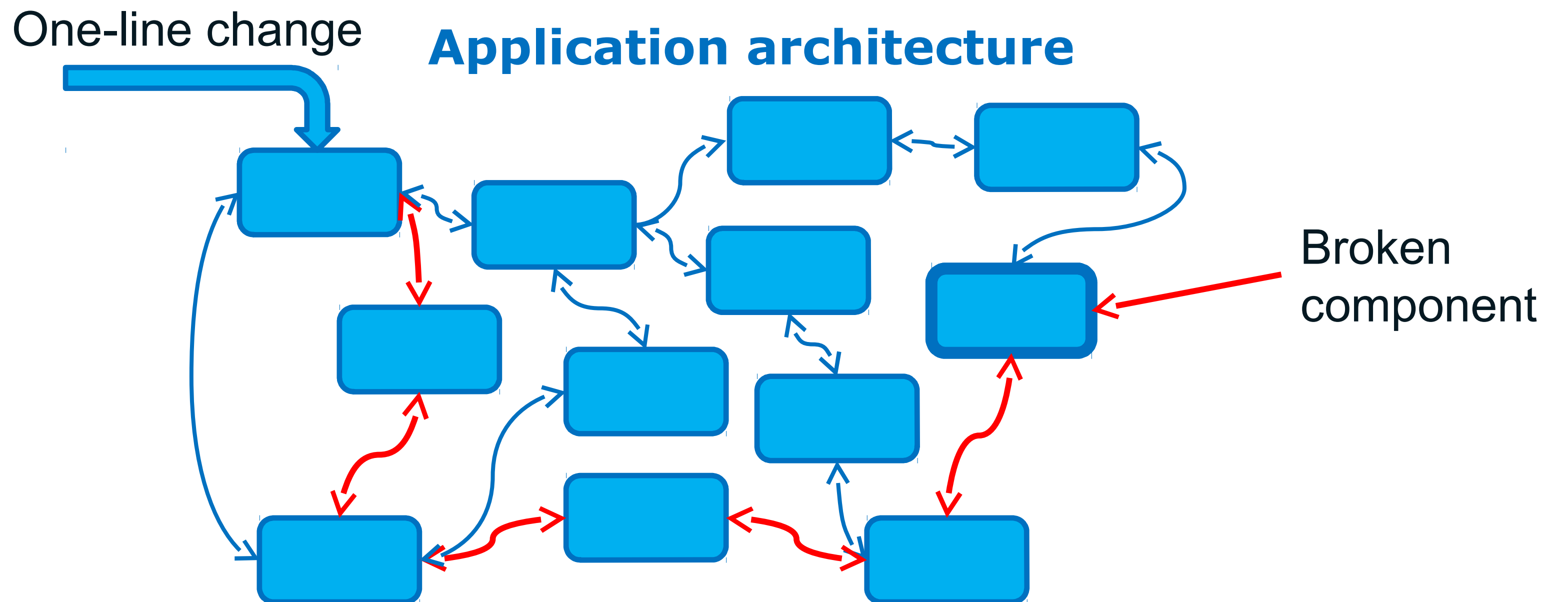
Optimization Notice

intel

Tests for a specific change-set

Select tests for the modified component

- Effective but dangerous: affected $\neq$ modified



Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Coverage-based test selection

- Concept:
- Start with a wide tests set covering all (or most) components
- **Exclude** tests that **do not cover** the modified parts of the program
- Classic approach:
- Collect coverage of basic blocks
- Build control flow graphs (CFGs) for original and modified programs
- Exclude the tests that don’t cover changes in the CFG
- Run the rest of test and update the code coverage

[Rothermel, G., Harrold, M.J. **A Safe, Efficient Regression Test Selection Technique**]

[Wong, W.E., Horgan, J. R., London, S. **A Study of Effective Regression Testing in Practice**]

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

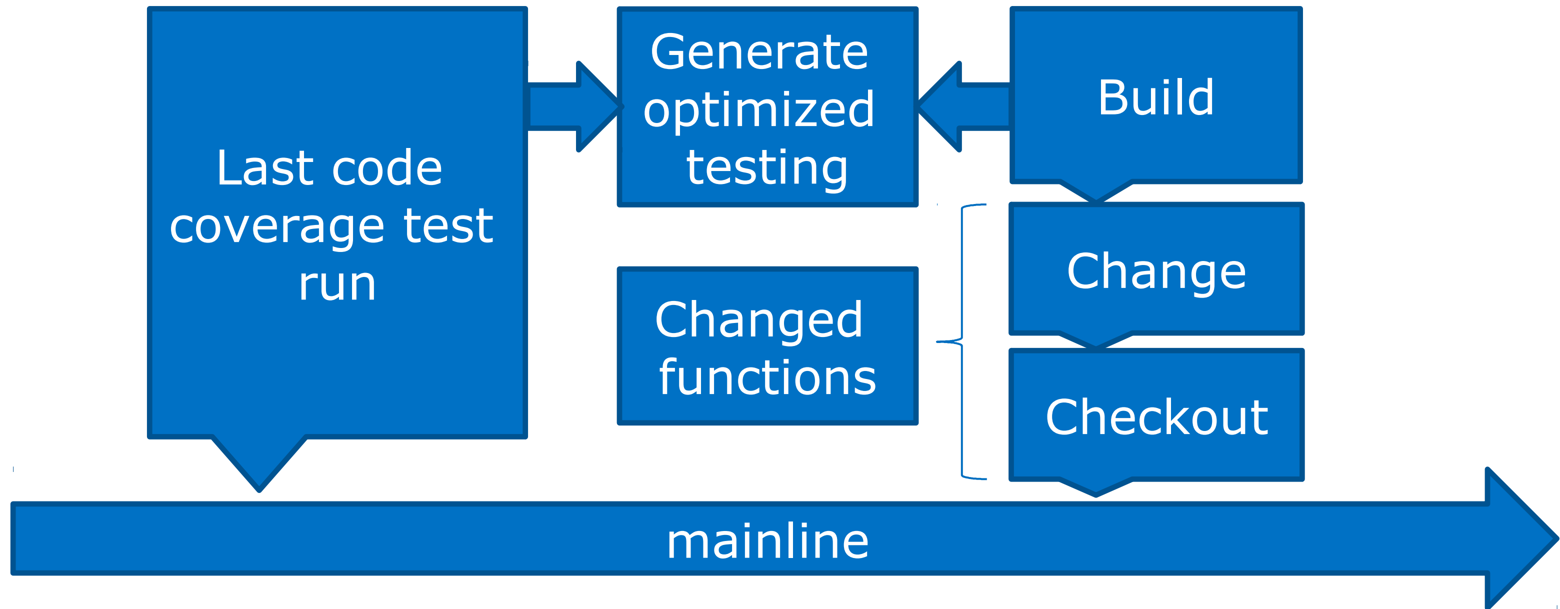
Optimization Notice

Issues with classic approach

- Basic block coverage
- For every test it requires lots of a disk space (in our case more than 1Tb per platform)
- For each test suite (merged) it’s still big and hard to analyze (more than 50Gb in our case)
- It changes very often and should be updated frequently (collection of coverage is 3 times longer than a regular testing)
- Building a complete CFG is expensive and redundant

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Code coverage optimized testing



Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Regular code coverage test runs

- Needed to collect information about existing testing coverage
- Result is mapping of source function names to a set of test suites/optsets that covers them (about 300Mb Perl hash file for each platform)
- Basic block coverage is not used for simplicity
- Performed once per week automatically
- Done on four platforms: Linux/Windows x32/x64
- Performed using specially build version of application with code coverage information generation enabled (-profgen Intel compiler switch)

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

intel

Specifics of using Intel compiler as test application

- After each individual test coverage data is merged to suite/optset to save disk space (still each platform requires about 50Gb to store it in a packed form)
- Coverage testing can take up to 6 days
- Special compiler wrappers are used to merge code coverage data after each 100th compilation to handle huge tests
- Tests are run in compile/link mode only but not all tests are designed for that thus some tests still execute and sometimes leave their code coverage too
- Not all platforms can be covered due to lack of testing pool resources

Optimization Notice

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

intel

Find changed functions

- Reused functionality of existing version control system
- Get list of modified sources
- Get original and modified version of sources
- For modified header files build dependency files are used to find dependent source files
- Build log is used to get correct compilation commands (macros and generated header files are important)
- Need some syntax parser solution to remove dependency on source formatting changes

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

Rejected source parsing solution

- Tried to use Python parser generated by ANTLR3 from already existing grammar
- Too slow (about 10 minutes to parse average source)
- Too much memory consumption (over 2GB per one modified/original source of average size)
- Need industry quality syntax parser

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

intel

Currently used parsing solution

- Based on compiler internal representation dumps (currently Intel compiler is used for that, also gcc -fdump-syntax-tree, GCC::TranslationUnit can be used for that)
- Global/routine symbol tables are already constructed in these dumps (so it is not just plain AST)
- Need to handle other ambiguity such as variable name suffixes, timestamp sensitive C++ mangling and __LINE__ type macros
- __LINE__/__DATE__/__TIME__ predefined macros are manually removed/restored in sources
- Multiple sources are dumped in parallel to save time

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Internal representation example

Routine IL0 dump

```
PACK| (i2.1) align: 0 MOD 4, size: 4, …
   VAR| (i2.1_V$1) type: SCALAR, size: 4,
        | offset: 0, esize: 4, …, edtype: SI32,  …
…
3    0        entry extern SI32  main
              {
4    1            i2.1_V$1 = 0(SI32);
5    2            return ( (SI32) i1_V$0 + i2.1_V$1 );
6    3            return ;
              }
Root Context C0.1 {
} C0.1
```

Module symtab dump

```
PACK| (i1) align: 0 MOD 4, size: 4,
        | …, offset: 0, …
   VAR| (i1_V$0) type: SCALAR, size:
4,
          | offset: 0, esize: 4, …,
          | edtype: SI32,  …
   INIT| offset: 0, repeat: 1, …,
          |data: 0(SI32)
```

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

Finding changed functions

- Each function that has changed internal compiler representation is considered to be changed
- Each variable usage it replaced by its basic name, type, align and other definition information, including complete initialization
- Changes in externally visible variable (or their initialization) are considered as a whole program change and code coverage testing optimization is not performed

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

intel

Issues with current parsing solution

- Need to demangle function names used in compiler dumps and code coverage runs since it was found that in some cases their mangling can differ
- Issues with not always reliable build dependence file information (some header dependent source names are missing in the build log files)
- Debug version of Intel should be used to get all required internal representation dumps
- Dumps can take up to 30Mb and can be generated several minutes in the worst case

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

Issues with overall approach

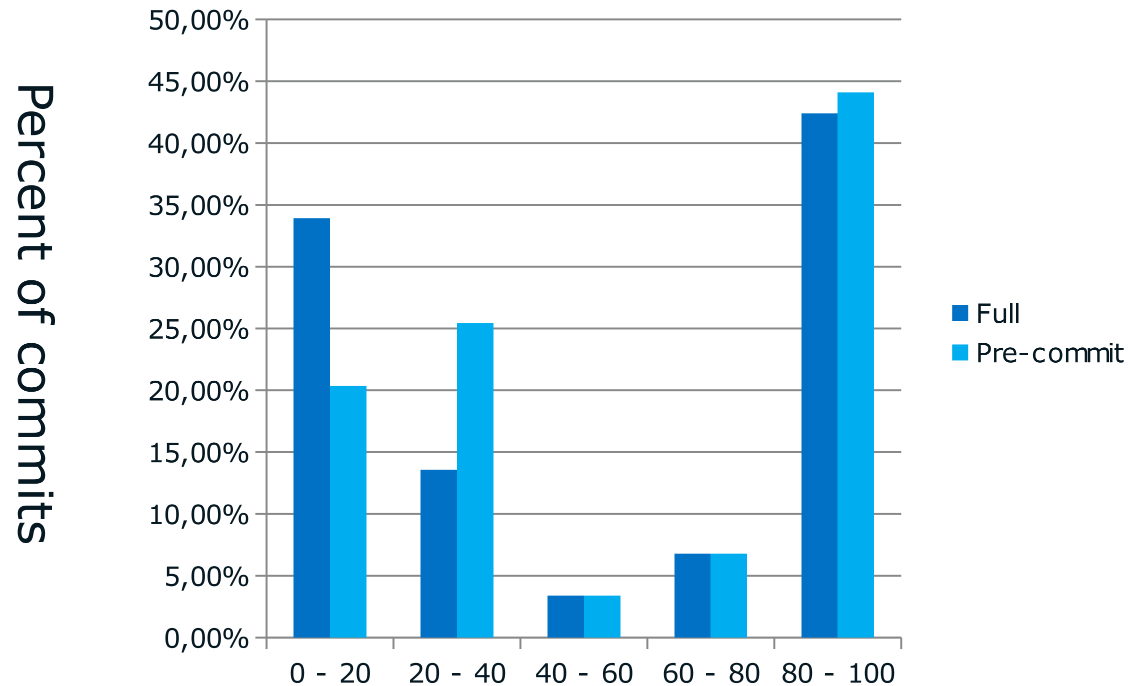
- Uses a week old (in a worst case) code coverage information
- Can’t adequately cover recent code
- Do not work so well with development braches (not mainline)
- Works well in case of small checkins, since component promotions:
- usually do not give significant testing reduction
- have high overhead in source dumping time (which adds to build time and is not paralleled on testing pool)

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

intel

Results



Percent of test suites selected for the testing

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice
intel

Conclusions

Amount of the pre-commit testing can be reduced considerably (55% on the average) even using less precise methods than classic ones

Overhead of the coverage-based technique can be significantly less than gain from shortened testing

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

intel

To the future adopters

- Keep your makefiles in order
- Correct prerequisite lists are needed to analyze the diffs
- Ideally incremental builds should work 100% correct
- Use code coverage on a regular basis
- It makes things easier if coverage test runs are set up already
- Coverage rate should be good for these methods to work properly

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

intel

2012 CEE-SEC(R)
Software Engineering Conference in Russia

Thank you!

Alexey Salmin
Alexander Stasenko

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

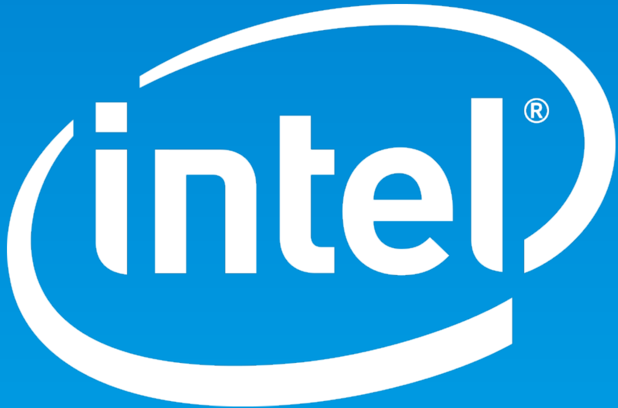
Copyright © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.


intel®