Collabora Productivity

Regressions: what, why and their extermination

Michael Meeks

General Manager at Collabora Productivity

michael.meeks@collabora.com

mmeeks, #libreoffice-dev, irc.freenode.net

“Stand at the crossroads and look; ask for the ancient paths, ask where the good way is, and walk in it, and you will find rest for your souls...” - Jeremiah 6:16

Overview

- Regressions:
  - What is a regression ?
  - How do they happen ?
  - What do we do to avoid them ?

- The future:
  - Improving unit testing
  - Better user testing:
    - Escaped regressions – the bad ones.

What is a regression ?

Regressions ...

It's now **broken !** – and it **used to work !**

- Escaped Regression:
  - This got released to an end-user somehow
    - – Who expected the suite to be stable ?
- Non-escaped
  - We trapped & nailed it before it got there.

Regressions – or not ?

- It still works, but it got 2x slower …
- Often a speed, memory, time trade-off in development:
  - If we make it 100x faster for one user's case.
  - Possibly it uses 2x as much memory for another user's case.
  - Perhaps it gets 2x slower for another user's case.
- So just revert the patch !
  - Now we get another “it got 100x slower” regression.
- My fix *is sometimes* Your regression ...

Two Antithetical Views:

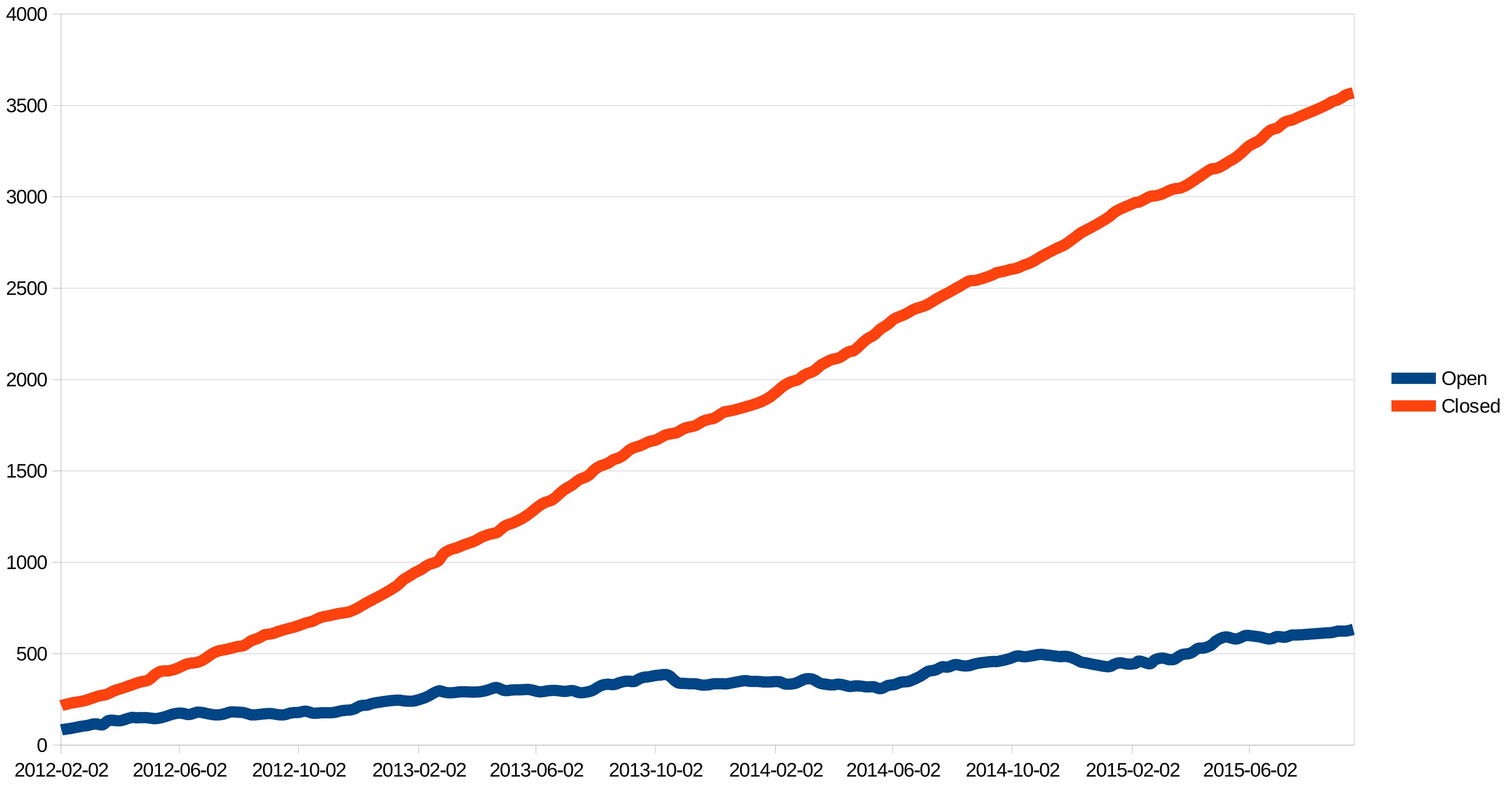
A. *"First you should fix all known bugs, then you can add features !"*

vs.

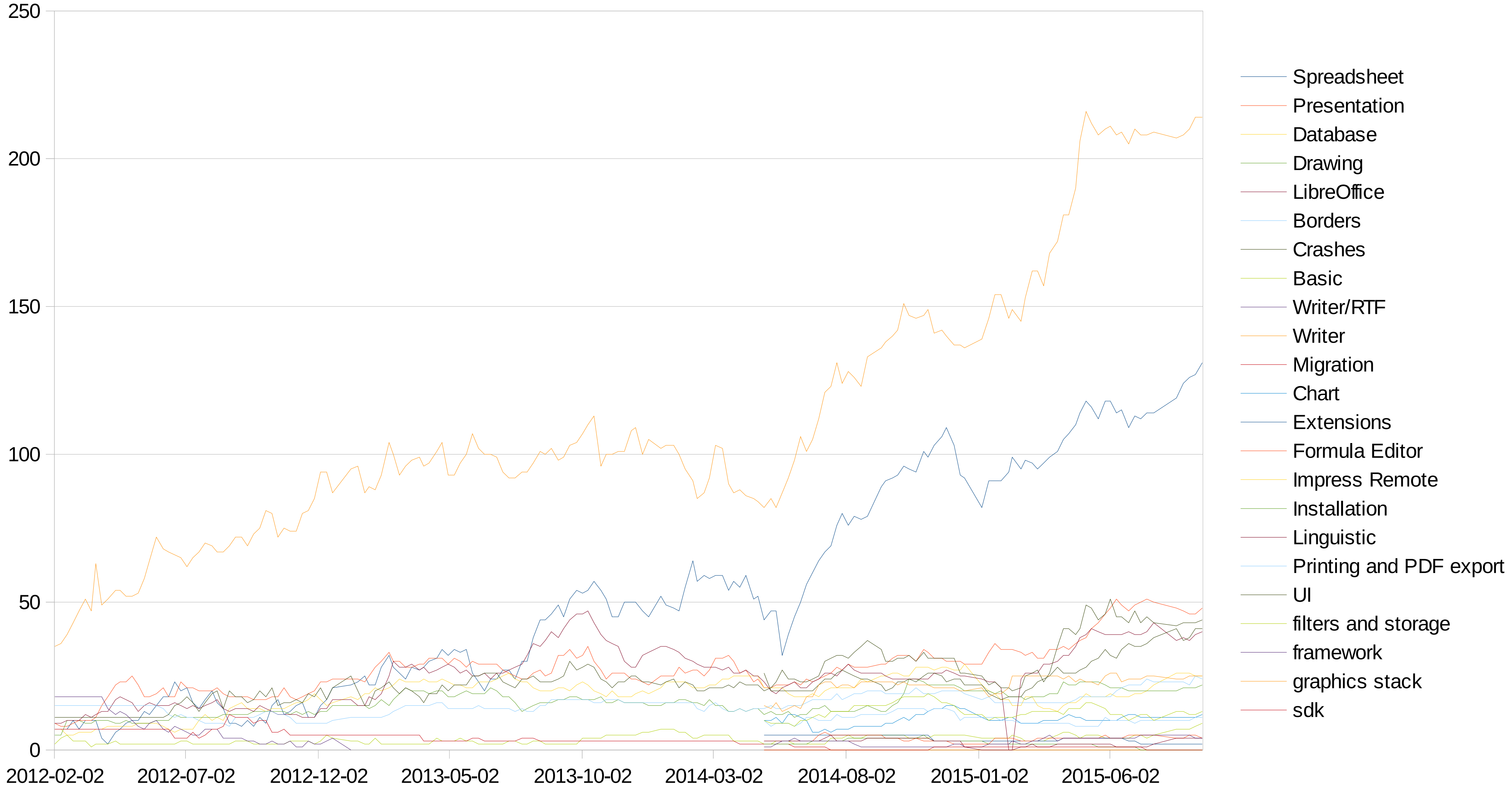
B. *"regressions are inevitable, I'll work on what I like, get used to it !"*

- My Goals:
  - **A** → *"given what is going on, perhaps we need to accept some regressions"*
  - **B** → *"perhaps we should invest more time in fixing and avoiding regressions"*

Regressions vs. Time ...



Regressions by component ...



How do they happen ?

Hard to avoid ...

- Sometimes have the privilege of interacting with people who are horrified by regressions:

  “It is my God Given right ! to never see a regression; any developer who creates one must be an idiot ! – we must find and stop them from committing, and have their code infinitely scrutinized”.

- I also don't know a developer who has never caused a regression:
  - in proportion to the degree of change.

Causes of regressions:

- Imperfect knowledge & understanding
  - Recall that there is no developer who has never created one of these.
  - It is not possible to know everything about 8 million lines of highly coupled legacy code.

Some Quick examples.

  - From 3500 fixed regressions – some quick thoughts … ask any developer for a truck-load more silly examples.

Regression examples:

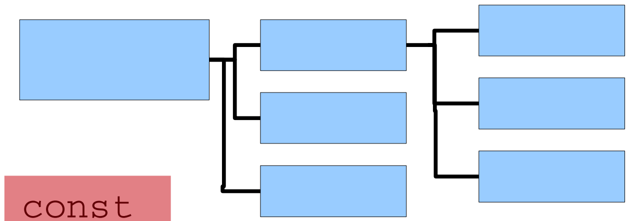
- Imperfect knowledge & understanding
  - fdo#61256 - the Get.*Export methods also create and register styles
  - A 'GetFoo' method should not have side-effects
  - We were loosing great chunks of style / attribute data on save.

```
--- a/xmloff/source/draw/sdxmlexp.cxx
+++ b/xmloff/source/draw/sdxmlexp.cxx
@@ -448,6 +448,8 @@ void SAL_CALL SdXMLExport::setSourceDocument( ...
         // construct PropertySetMapper
         UniReference < XMLPropertySetMapper > xMapper = new
XMLShapePropertySetMapper( aFactoryRef);

+        // get or create text paragraph export
+        GetTextParagraphExport();
         mpPropertySetMapper = new XMLShapeExportPropertyMapper( xMapper, *this );
         // set lock to avoid deletion
         mpPropertySetMapper->acquire();
```

Defensive coding has limits:

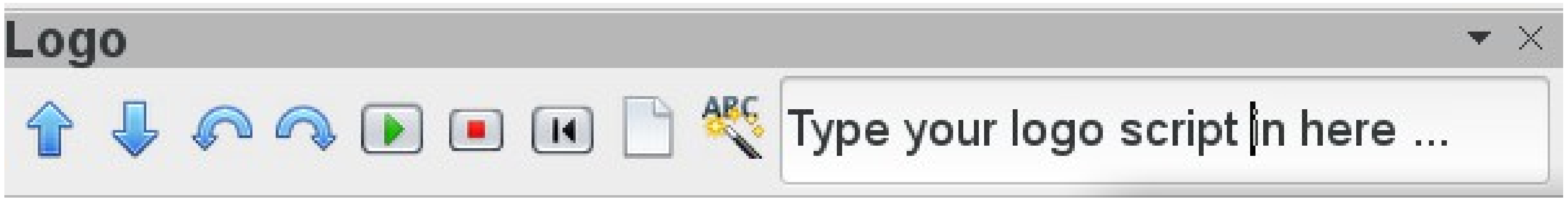
- VCL: has a beautiful tree of windows:
- Various methods walk over the tree doing things eg.



```
2004-07-06 Size ToolBox::CalcMinimumWindowSizePixel() const
2004-07-06 {
2004-07-06     if( ImplIsFloatingMode() )
2004-07-06         return ImplCalcSize( this, mnFloatLines );
2004-07-06     else
2004-07-06     {
2004-07-06         // create dummy toolbox for measurements
2015-03-30         VclPtrInstance< ToolBox > pToolBox( GetParent(), GetStyle() );
2004-07-06
2004-07-06         // copy until first useful item
2004-07-06         std::vector< ImplToolItem >::iterator it = mpData-
>m_aItems.begin();
2004-07-06         while( it != mpData->m_aItems.end() )
2004-07-06         {
2004-07-06             pToolBox->CopyItem( *this, it->mnId );
...
2004-07-06         // add to docking manager if required to obtain a drag area
2004-07-06         //  (which is accounted for in calcwindowsizepixel)
```

Unexpected side-effects:

- LibreLogo / PyUno goodness …



- Wonderful feature – good for kids & schools.
- Who would predict that adding this would impact startup time severely ?
  - even without the toolbar being visible ?
- High-quality image scaling from 26x26 → 24x24 on every startup – before showing the UI.
- Lame framework code:
  - In general good for programmers to be lazy though.
  - Don't optimize for cases that don't happen: until they do.

Bug fixes cause regressions:

tolerate pngs with invalid tailing IDAT chunk lengths

```
 PNGReaderImpl::ReadNextChunk()

 mrPNGStream >> mnChunkLen >> mnChunkType;

 rChunkData.nType = mnChunkType;

- // #128377#/#149343# sanity check for chunk length

- if( mnChunkLen < 0 )

- return false;

+ // fdo#61847 truncate over-long, trailing chunks

const sal_Size nStreamPos = mrPNGStream.Tell();

- if( nStreamPos + mnChunkLen >= mnStreamSize )

- return false;

+ if( mnChunkLen < 0 || nStreamPos + mnChunkLen >= mnStreamSize )

+ mnChunkLen = mnStreamSize - nStreamPos;
```

Very badly formed PNG image … data chunk has a completely bogus length: vcl/source/gdi/pngread.cxx

Strategies to avoid regressions:

Strategies to avoid regression:

- Quality through obsolesence
  - If change causes regressions: don't change anything !
    - *It works ! A truly effective way to avoid regressions.*
  - Define the datum as 'now: and bingo no bugs.
    - All those old bugs turn into much-loved and understood 'features'.
- Problem is:
  - People's perception of quality:
    - not conditioned by 'bugs' – but also by 'bugs' like:
      - “My document doesn't open”, or “My document doesn't render”
  - “but those are new filter / layout features” - right ?
- We have to build a community → people need to see their changes.

Strategies to avoid regression:

- *“Stop-world to just fix bugs”*
  - Don't allow any commit unless it only fixes a bug !
    - Lets do this for six months and then we'll have wonderful quality !
- Actually **we already do this**:
  - Our release schedule has stable branches
  - Enterprises can buy “long term stable” branches
    - You can have 3+ years of bug-fixing.
- Problem is resource mgmt. most people who say the above mean:
  - *“You should first fix my bug and then you can get on adding features I want” =)*
- Bug 'fixing' can also cause regressions ...

Strategies: the obvious ones

- Compiler warnings
  - We're warning-free on all major platforms
  - We go turning on awkward warnings.
- Static code-checking:
  - CppCheck → lots of commits.
  - Coverity → Zero score
  - Clang plugins → lots to catch mis-use of various problematic APIs & patterns.

Strategies: human testing ...

- Human beings test LibreOffice
  - Mostly on the triage / daily use side.
- Every escaped regression is a compound failure:
  - A developer caused it
  - All users failed to test pre-release builds & report it.
- Ideal testing is alongside the developer:
  - During feature development …
- Most encouraged during VclPtr to see the breadth of testers of pre-releases; using strange features.

Strategies: human bisectig ...

- Finding the right person to blame
  - bibisection is really important
    - An awesome productivity tool.
- Important to close the cycle quickly:
  - If someone has a change that creates lots of bugs; they need to know fast & look at fixing them.

Strategies: Unit testing

- Michael Stahl's talk → comprehensive view.
- If you commit just a bug-fix
  - You will get to fix it again later
    - First soon, and second much later.
- If you commit a unit-test as well
  - Someone else gets to improve their fix.
  - Your fix stays fixed.
- CppUnit
  - 200 discrete test modules
  - ~3500 tests
  - 16000 assertions etc.

Strategies: Test Automation

- Doing testing very regularly
  - during every developer compile – run tests.
- Jenkins / Continuious Integration
  - Integration testing for (many) check-ins
- Tinderboxen:
  - Loop-building with extra clang static checking, running the tests, on many platforms.

Strategies: Test Automation #2

- Crash-testing:
  - 75,000+ documents:
    - Load , save , validate testing → **~daily** …
  - Do that again with Address Sanitizer, valgrind
- Performance Testing
  - Hard profiling under callgrind CPU simulator
  - Catching performance regressions …
- Testing all old security CVE documents ...

Strategies: Code Review ...

- Mandate code-review for all patches
  - extremely expensive in developer & reviewer time.
  - We reserve this for:
    - new contributors
    - stable branches
      - double review for bug fixes
      - tripple review for new features.
    - voluntary input for wise / scared developers
      - please check this change to a dodgy bit of code.
  - Unfortunately – too little review bandwidth.
- Hoping to improve this by TDF funding new contributor reviews → Mentoring lead.

An example: VclPtr …

VclPtr change ...

- Intended to be -minimal- not a complete fix, but getting the simple, basics in-place, avoid touching the tar-baby too hard etc.
  - (its the way you tell them)
- VclPtr tracker bug (zero open)
  - As of today: 61 regression bugs tracked.
    - Great work from the QA team.
    - Left paranoid assertions on – no longer needed.
  - 5 bugs 'escaped'
    - 5.0.1 → 3 fixed.   5.0.2 → 2 fixed.

VclPtr: retrospective ...

- Missing anything to just open & close all dialogs.
  - mjayfrancis → working on this one with beautiful pyuno / accessibility code.

The Future:

Improving the situation ...

- Killing the areas where it is *hard* to write the first / a new unit test:
- ESC / lots of ideas on writing unit tests.

  A. Cross-platform font/shaping stubs for layout tests

  G. Improved Format Validity Checks

  B. Automated Help/Documentation screenshot creation

  D. Automated a11y based UI testing

  C. SSDs for prominent QA developers

  H. Always Green Master

  E. Checking for DSO dependencies

  F. Android Unit Testing

Conclusions

- Mission: Make LibreOffice Rock
- Quality is an important part of that
- Regressions bite unpleasantly
- Finding & nailing them in advance is possible
- Automated tooling makes this much easier ...

Oh, that my words were recorded, that they were written on a scroll, that they were inscribed with an iron tool on lead, or engraved in rock for ever! I know that my Redeemer lives, and that in the end he will stand upon the earth. And though this body has been destroyed yet in my flesh I will see God, I myself will see him, with my own eyes - I and not another. How my heart yearns within me. - Job 19: 23-27