Reducing technical debt and complexity by promoting collaborations

Yann Pouillon
Universidad de Cantabria, Santander, Spain

2019/05/20
9th international ABINIT developer workshop, Louvain-la-Neuve, Belgium

Acknowledgments

Cliparts downloaded from http://clipart-library.com/

Outline

- What is technical debt?
- Examples
- Collaboration is prevention

What is technical debt?

Quick summary of technical debt

Technical debt is a normal byproduct of ongoing developments.

- Avoiding work today by promising to do it tomorrow
- Trade-off: benefit(getting it now) > burden(fixing later)

- Types of technical debt
  - Deliberate: strategic/tactical choice (must track)
  - Accidental: implementation reveals flaws
  - Bit rot: complexity from outdated design

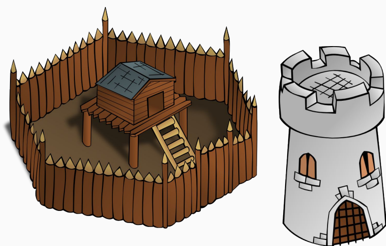
- What it is not
  - Procrastination
  - Bad programming practices (= unsustainable)
  - Failure of planning (= planning for failure)
  - “Rush-to-keyboard” syndrome

- How to avoid it
  - Design first
  - Refactor periodically (e.g. every other year)
  - Test-Driven Development

Typical sources of technical debt

What got you here won't get you there.

- Evolving understanding
  - Design patterns, architecture, standards
  - Code reviews, pair programming
- Change in context
  - Languages: F90 → F2003 → C++14, Py2 → Py3
  - Processes & tools: Bazaar → Git+Gitlab
  - Philosophy: silos, monoliths → shared modules
  - Availability: careers, single points of failure
- Deviation from original purpose
  - Fallbacks: MPI-IO ⇒ HDF5 complexity



2000

2019

Impact and remediation

- Which area does the technical debt impact?
- How intense / extended is the impact?
- Who does it affect?
- Does it prevent other efforts?
- Does it hinder collaboration?
- How urgent is it to remediate?
- What is the first step to remediate it?

Examples of technical debt

Example: abilint

From an asset to a road block.

- Why?
  - Fortran: no automatic dependencies
  - No explicit Fortran interfaces
    ⇒ segfaults, unpredictability
- 2005-2018: abilint for call graph + interfaces
  - Machine changes in versioned files
  - No Fortran 2003 support
  - Complex and single-threaded
- Evolution of ABINIT
  - 400 klines ↝ 1100 klines
  - 15 ↝ 40 contributors
  - Procedural programming ↝ OOP

Example: Bazaar

When the context kills the “VCS for human beings”.

- Why?
  - CVS unsuitable, Subversion insufficient
  - Need for Distributed Version Control
  - Bazaar: user-friendly, easy to learn
- 2004-2016: Bazaar for Version Control
  - Still immature when adopted
  - 2005-2007: C → Python 2 → GNU Project
  - 2014-2016: end of story
- All factors external to ABINIT
  - Private funding of Bazaar
  - Development within a single company
  - Considered as a project, not a product

Example: ABINIT Fallbacks

From a quick fix to an infrastructure component.

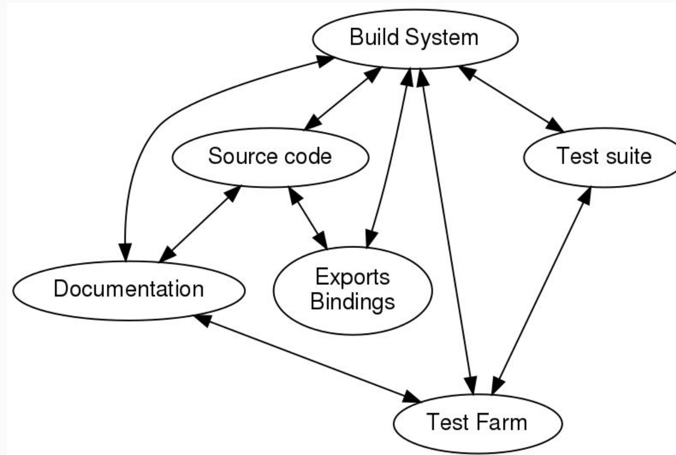
- Why?
  - ABINIT shipped with dependencies
  - Fortran modules ⇒ binary incompatibilities
  - Developers need help
- 2005-2014: “temporary” ABINIT component
  - Minimalistic, consistent set of versions
  - Heterogeneous: C, F90, F95, F2003
  - Individual test suites not run
  - Breaking feedback loops
- 2014-2019: standalone package
  - Let build system support alternatives
  - Large deviation from original purpose
  - Single-point-of-failure removal

Technical debt in the build system

User interface of the build system

How to stabilize the UI while adjusting to evolving specifications?

- Automatic makefile generation
  - Design of abinit.src: before ConfigParser
  - Executed Python = security vulnerability
- Oriented on human error correction, but
  - Proliferation of automated frameworks
  - Complexity of dependencies (e.g. linalg)
- Interactions between components ⇒ team work



Circular dependency

ABINIT depends on BigDFT which depends on … ABINIT.

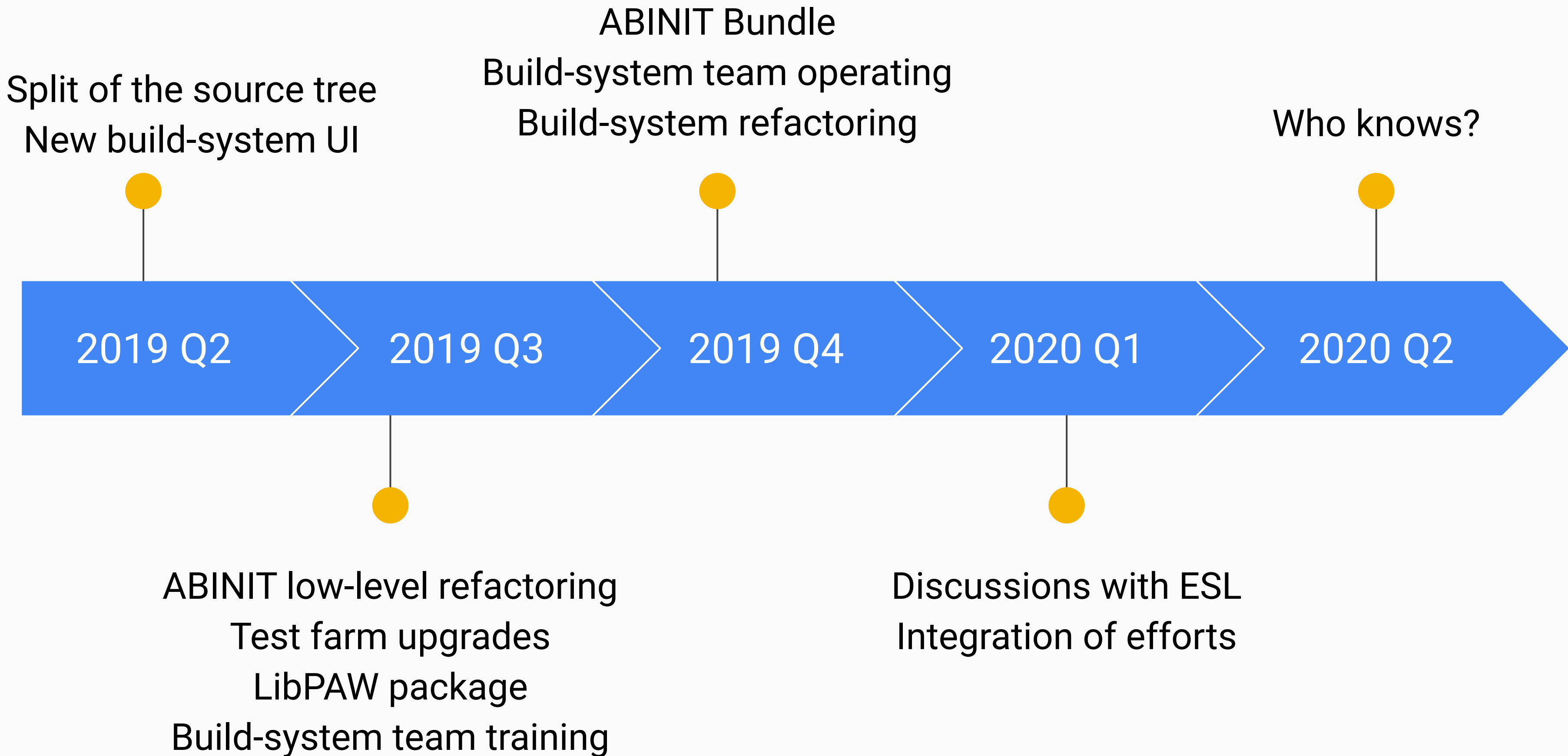
- 2009: BigDFT internal copy of ABINIT low-level
    - Affects fallbacks, build system, source
    - Complexity from namespace clashing
- Social cause ⇒ solution not only technical
- Stage 1: rename ABINIT low level (2011-2015)
- Stage 2: maintain patched BigDFT (2013-2019)
- Stage 3: split ABINIT source tree (2017-2019)
- Stage 4: restructure build system (2016-2019)
- Stage 5: organize ABINIT-BigDFT collaboration

Fortran

The Fortran Standard Committee does not address core standard issues.

- Main issue: Fortran modules not in standard
  - Incompatible between compiler vendors
  - Incompatible between compiler versions
  - Undefined behaviours with nested deps
  - Full automation impossible
- Since 2015: little evolution of standard
  - Main focus: interoperability with C
  - Vendors do not bother going to meetings
- Since 2017: the beginning of the end?
  - Developers switching to C++
  - Parallelization around Python 3
  - Fortran openly hated by young researchers
- Build system: Fortran = #1 complexity source

Rolling-wave roadmap


ABINIT Bundle
Build-system team operating
Build-system refactoring
Split of the source tree
New build-system UI
Who knows?
2019 Q2
2019 Q3
2019 Q4
2020 Q1
2020 Q2
ABINIT low-level refactoring
Test farm upgrades
LibPAW package
Build-system team training
Discussions with ESL
Integration of efforts

Thank you!