Development tools: Past, Present and Future

Marshall Clow, C++Now 2024

mtclow@gmail.com

We want tools that will help us write better software more efficiently

A long time ago…

The 1970 -> 1980s

The 1970s -> 1980s (1)

- Programmers Workbench (Bell Labs, 1977)
- Visual editors
- Linkers
- Make (1976)
- Source control systems (SCCS, RCS, CVS, etc)
- Manual implementations of virtual memory (for code and/or data)

The 1970s -> 1980s (2)

- Integrated development environments
  - Editor, compiler, linker all packaged together into one program
  - Turbo C, MPW, Think Pascal
  - Later: Xcode, Visual Studio, CLion, Cevelop, and many more
  - Manage multiple file projects
  - Manage configurations (debug, release)
  - Language-aware editors
    - Jump between definition and declaration
    - Syntax coloring

Then what happened?

CPU cycles became *much* cheaper

- CPUs got faster
- Many execution units in a single core
- Many cores in a single computer.
- RAM/storage got cheaper, too
- Now we have “Compute Farms”, where Amazon/Google/Microsoft/others will sell you huge amounts of computing power.

Networking became ubiquitous

- All computers come with networking built-in
- Networks got faster
- Online software distribution took off
- Moving data to/from the compute farms was simple, which made them more valuable
- Source control systems made teams writing software more efficient

GCC took a wrong turn

Around 2000 or so, Richard Stallman said (paraphrased):

“GCC is a compiler, not a library”

Process Improvements

- Test-driven development (1999)
- Agile development (2001)

Test-driven development

- Have automated tests
  - Run them often
  - Keep them passing
- Develop the tests while developing the code
- When you get a bug report, first write a test that displays the bug, and add that to your test suite.
- If the tests all pass, your code is good enough to ship.
  - If your code is not good enough to ship, improve your tests.

Agile Development

- Builds upon the TDD ideas
- Incremental development
- Always have a running system
- Short development projects (sprints)
- Lots of communication between team members and customers
    - “Stand-up” meetings

The last decade

The last decade (1)

- Sanitizers
- Fuzzing
- Git (2005)
- Github
- Clang-tidy
- clangd

The last decade (2)

- Godbolt
- C++ Insights
- “Time travel” for debugging
- Continuous Integration
- Configuration management (docker, apt, homebrew)
- Formal Method Tools

The last decade (3) - AI

- Write some code
- Write tests
- Summarize code
- Improve code
- Decipher compiler error messages (cwhy, CLion)
- Suggest fixes to failing programs (Chat-DBG)

The Future

“Prediction is very difficult,
especially about the future”

— Niels Bohr

The future - 1

The obvious bits

- CPU cycles will continue to get cheaper
- RAM/storage will get larger
- Connectivity will get faster
- Every new advance in software will be heralded as “AI”

The future 2

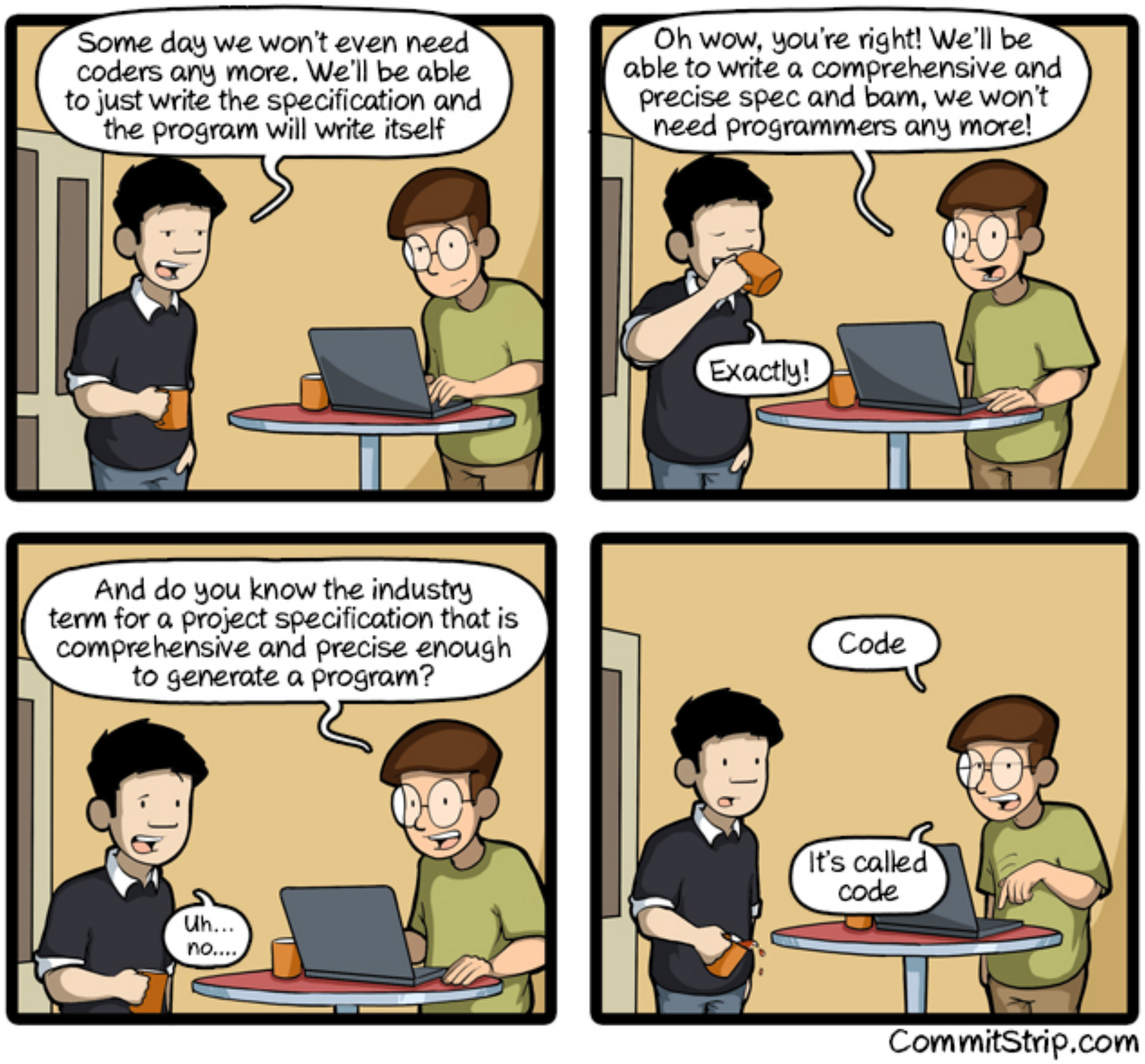
Tooling

- Static analysis will continue to get better
- Lifetime analysis will continue to improve
- Language-aware tooling (editors, etc) will continue to improve
- Formal methods will become easier to use.

The future - 3

More and more tasks will be automated by “AI”

- Larger and larger tasks
- More and more tasks will be assisted/automated (‘cwhy’ is a great example of this)
- Writing good prompts will become an important skill, like knowing how to write good search queries.
- Things I haven’t thought of…


Some day we won't even need coders any more. We'll be able to just write the specification and the program will write itself
Oh wow, you're right! We'll be able to write a comprehensive and precise spec and bam, we won't need programmers any more!
Exactly!
And do you know the industry term for a project specification that is comprehensive and precise enough to generate a program?
Uh... no....
Code
It's called code
CommitStrip.com

The future - 4

“Prepare a pull request against https://github.com/boostorg/unordered implementing the transparent try_emplace methods specified in https://wg21.link/p2363. Include tests.”

“A customer reports that the layout of the master report is messed up when it is printed on A4 paper. Prepare a pull request against XXXXX to fix this.”

Things I haven’t thought of - a senior project

Programming in Python, Stoia has written script that scrapes job advertisements off LinkedIn. Her script identifies key words within the job posting, which she uses to tailor a resume with ChatGPT and Google Bard. This helps the resume stand out through the software many hiring managers use to process and filter applications.

https://www.lakeforest.edu/news-and-events/seniors-thesis-investigates-how-ai-can-help-secure-interviews

A bifurcation in software development?

One-off software

- Software written to solve a problem, once.
- Once it has run, it is (frequently) discarded.
- Examples:
  - Gathering data/charts for news articles/blog posts.
  - School assignments
  - Data conversions
  - Source code refactoring projects

Long lived-software

- Software written to solve ongoing problems
- Runs often, if not continuously
- Examples
  - Libraries
  - Developer tools
  - Business process software
  - Process control software
  - Many others

Thank you

Questions?