The Proof Must Go On

Formal Methods in the Theater of Secure Software Development of the Future

Charles Averill

October 2025

University of Texas at Dallas, Dartmouth College

Trust in Software
(the concept, not an imperative)

What is trust?

- Confidence software behaves as intended
- Reliability, security, correctness
- Trust varies by user and context
- Often invisible until broken

Who needs to trust software?

- Users of critical systems (finance, healthcare)
- Governments and regulators
- Developers building on existing code
- Society at large relies on infrastructure

What are formal methods?

- Mathematical verification of program correctness
- Proofs, model checking, type systems
- Mostly academic until recently
- Reduce bugs and security vulnerabilities

Pioneering

The Early Days

1920s-50s:

- Hilbert's Program spurs research into fundamental questions
- What is it possible to know? And under what conditions?
- Leads to invention/discovery of foundations of formal programming languages

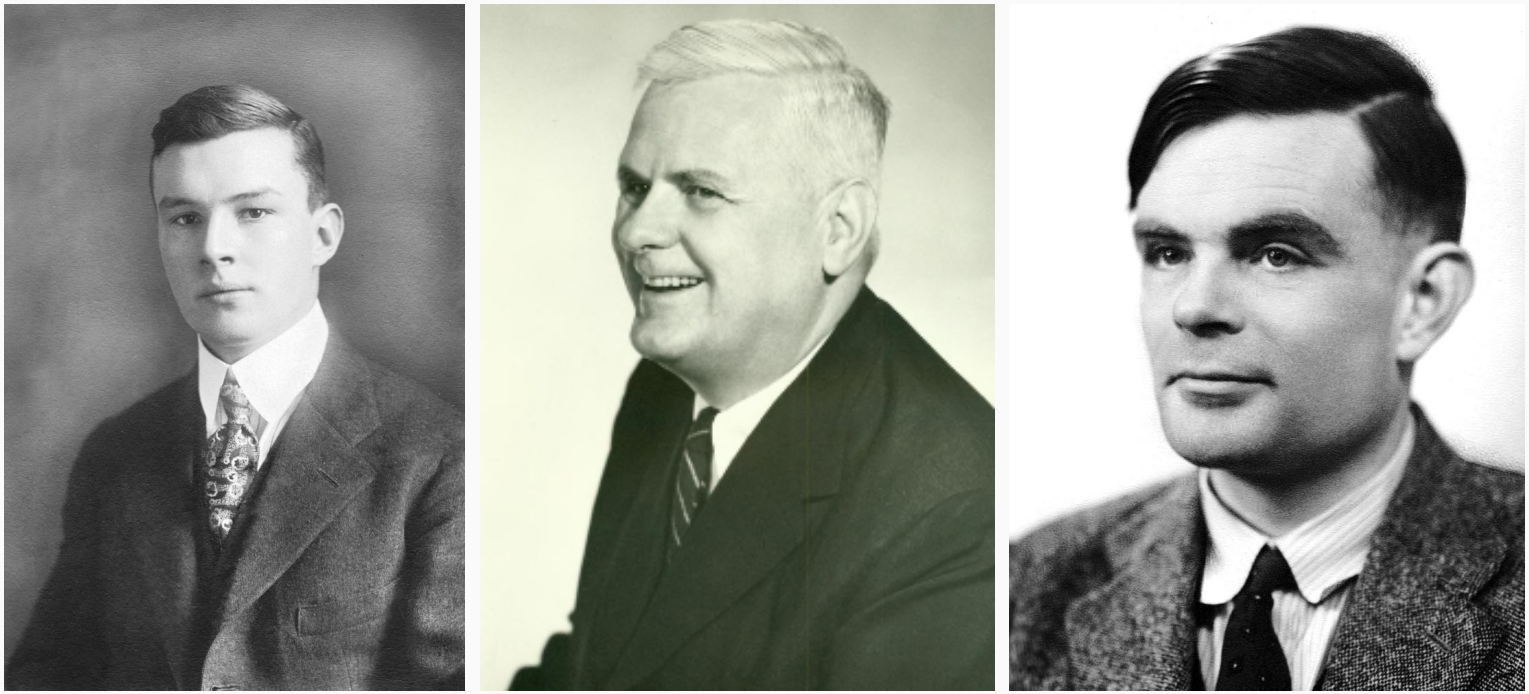


Figure 1: Curry, Church, Turing

The Early Days

- First electronic computers are developed for military purposes (so expensive!)
- Scientific/Business computers created post-WW2

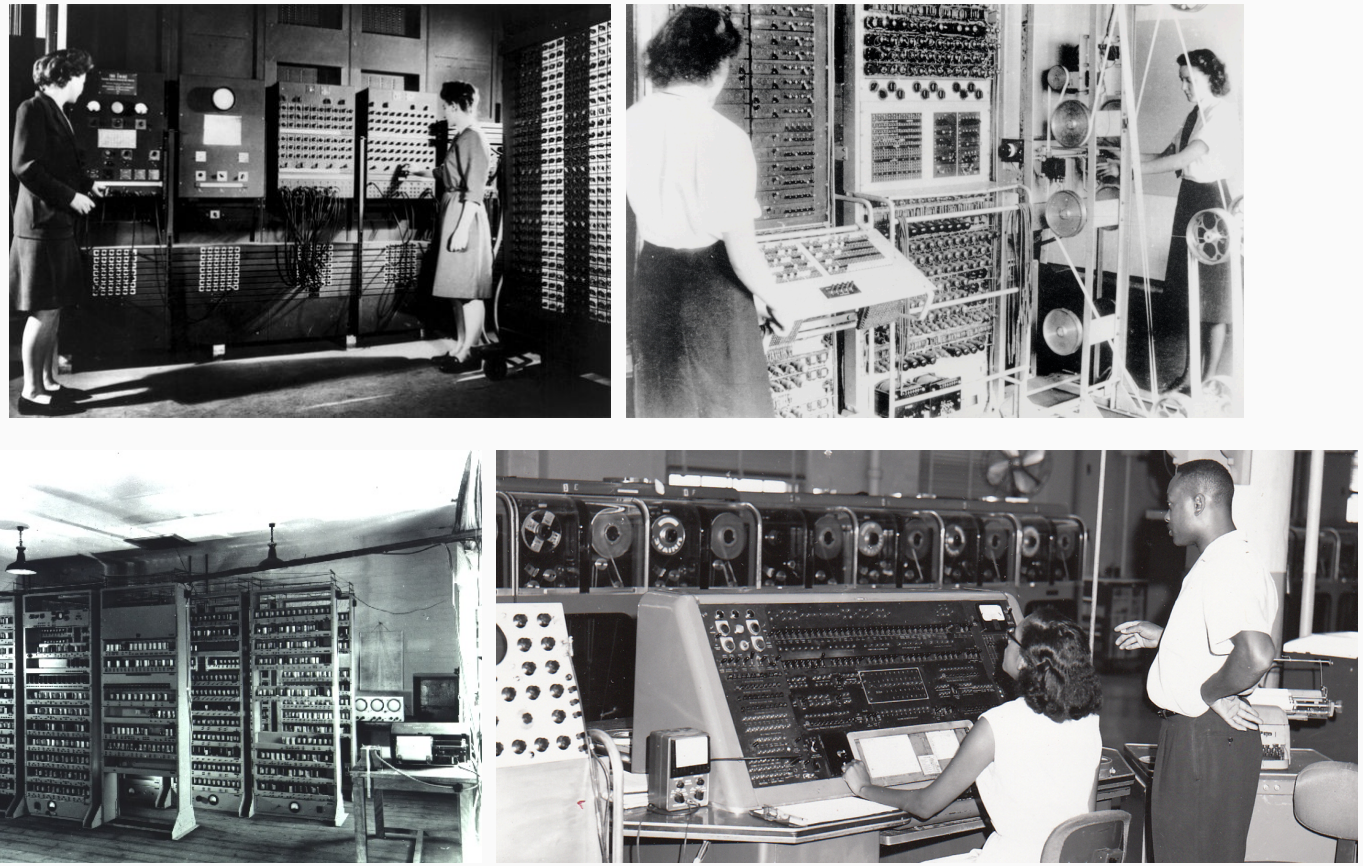


Figure 2: ENIAC, Colossus, EDSAC, UNIVAC 1

The Software Crisis

- 1960s: Hardware costs drop, but software costs soar
- **Problem**: how do we stop developers from writing lots of low-quality, unmaintainable code?
- **Solution**: robust reasoning tools for program analysis

The Software Crisis

- **Goal**: build reasoning tools in order to develop better software
- Denotational semantics introduced
- Hoare Logic, weakest precondition inference, “Assigning Meaning to Programs”

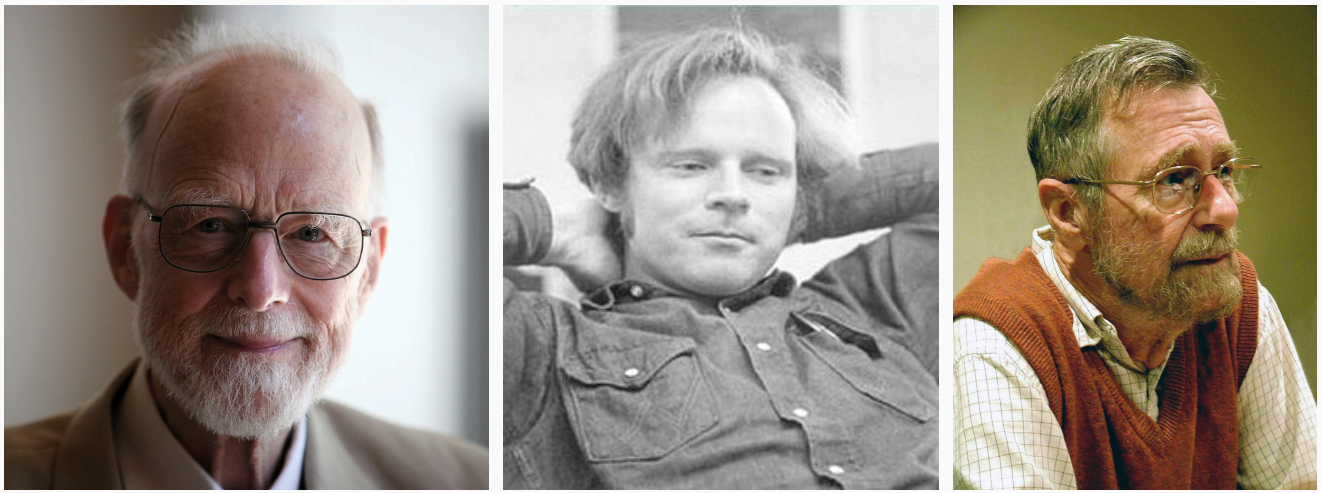
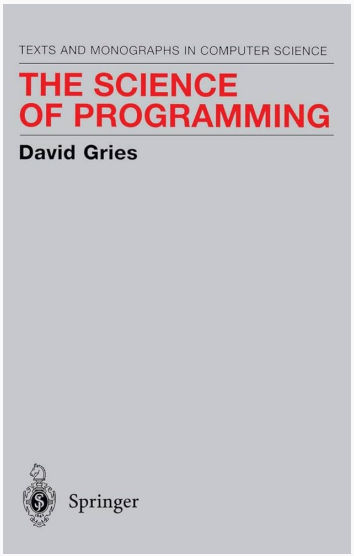
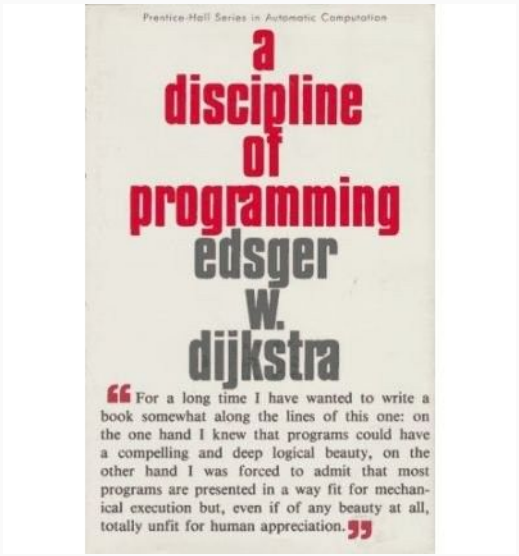


Figure 3: Sir Tony Hoare, Robert Floyd, Edsger Dijkstra

Hitting the Books

- By 1970s/80s, FM is common in standard university curricula
- Students required to develop program analysis skills
- Generations educated on prior 20 years of research





Giving it to The Man

- Programming practices influenced by defense needs
- Government programs: Cold War funding, Ada
- Type systems, model checking, process calculi developed

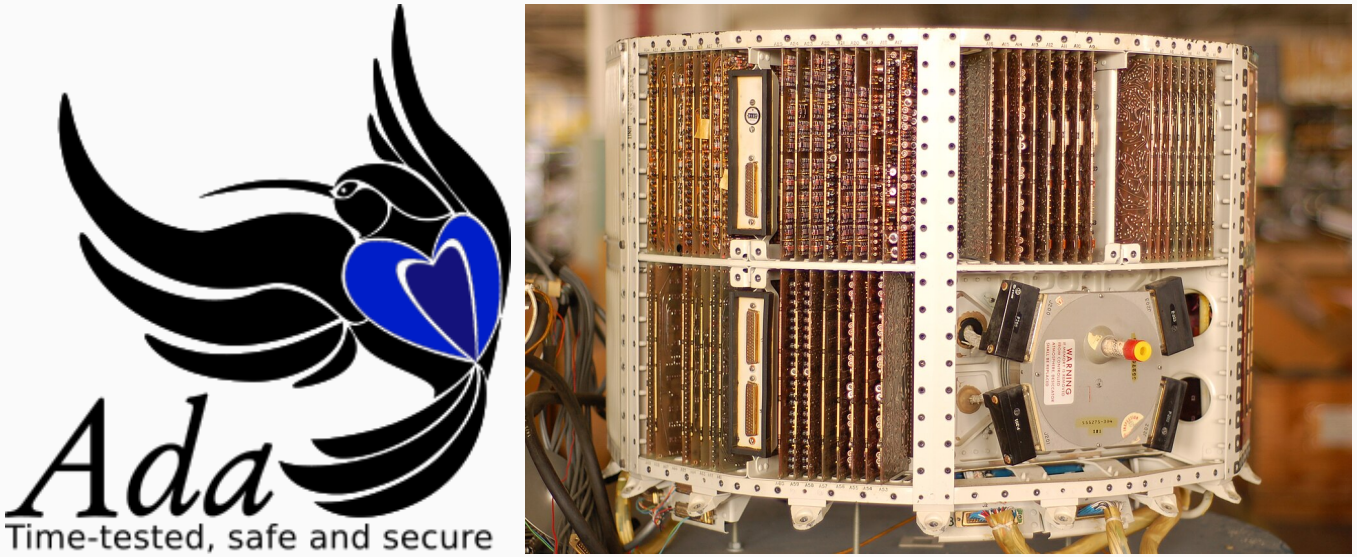


Figure 4: Ada, D-17B Missile Guidance System

The New Century

The Divergence

- Two (rough) camps of program analysis: formal methods and programming techniques
- Programming methods and techniques becoming widely adopted: unit testing frameworks, smarter IDEs, software model checking
- FM progresses in the background: separation logic, certified compilers, SMT-COMP

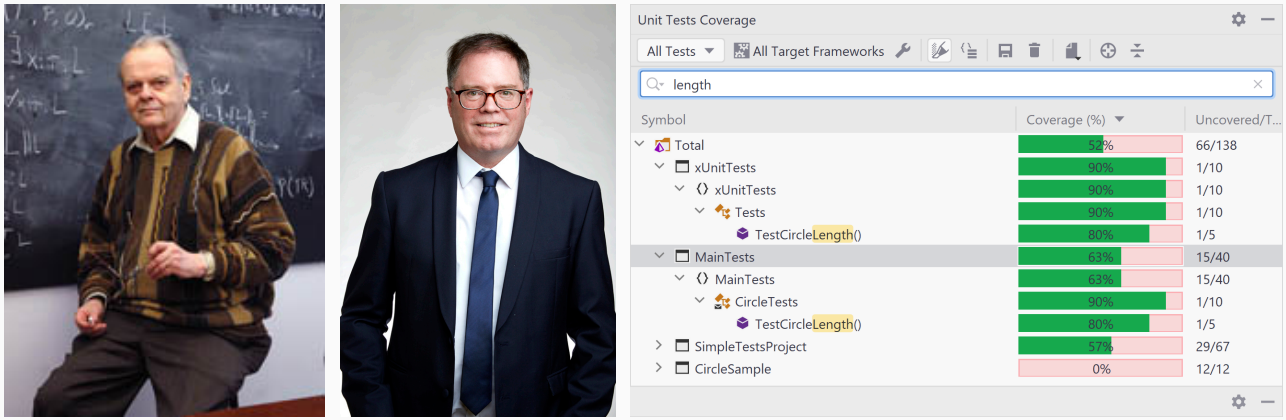


Figure 5: Reynolds, O'Hearn, unit test coverage tool

Adoption

- Governments, military, aviation, supply chains adopt many forms of FM in greater quantities
- Cloud and HFT companies adopt verification
- Verified components enter production gradually
- Early industrial successes build confidence

Back to the Present

Where are we now?

- Wealth of tools for all purposes at our disposal
- Uneven adoption due to high costs of training developers, rewriting software
- Adoption being driven by pressure from cybercrime
- FM courses slowly return to undergrad programs as electives

The Pattern

Academia

- Most FM discoveries come from universities
- Motivated by solving current problems of the day
- Hilbert's Program, code quality, code security
- Research often years ahead of industrial use

Government

- Funds large-scale research and applies it to critical systems
- Takes academic ideas and sponsors iteration and application
- WW2 cipher-breaking, Cold War systems, drones, computer infrastructure
- Often the only source of funding large enough to take risks on unproven research

Industry

- Adopts proven techniques from government and academia
- Adoption motivated by profitability or infrastructure needs
- Typically waits for validated, large-scale results
- Exception: massive companies can fund some research themselves

Education

- Students given opportunities to study cutting-edge research, but not required
- Curriculum focuses on marketable skills first
- Writing correct software, OOP, modern programming practices
- Prepares next generation to adopt new methods when they reach industry

Summary

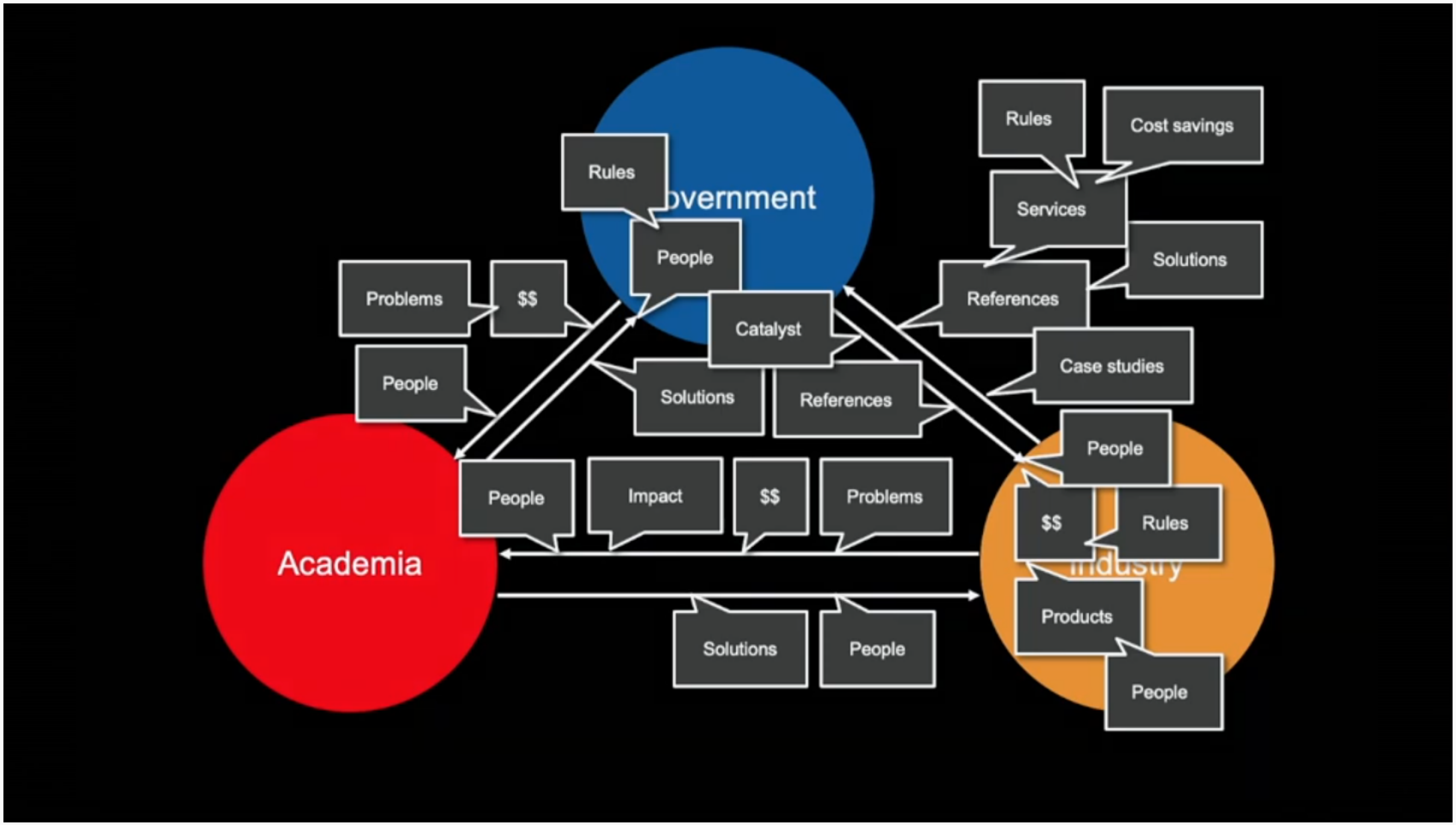


Figure 6: From “Thoughts on the interplay between corporate, government, and university R&D” - Byron Cook [1]

The Secure Software of Tomorrow

Timeline

- Hybrid top-down/bottom-up verification
- MVCs, verified libraries, language support
- Dealing with hardware
- Educational reforms driven by cybersecurity needs
- Deus ex Machina

Meeting in the Middle

- Combine high-level semantics and low-level effects
- Full-stack proof frontends emerge
- Reason about correctness, processor, timing, cache
- Enables more complete software verification

Minimal Verified Components

- Small, verified building blocks to replace critical low-level code
- DARPA-funded development
- Rapidly proliferate across libraries
- Parsers, encoders, cryptography

Veritas

- First verified C standard library
- Uses MVCs extensively
- Compilers allow substitution in standard libraries
- Adoption spreads to embedded and open-source code

The Unverifiable Core

- AI and I/O partially unverifiable
- Hybrid trust strategies required
- Hardware specs formalized by GIANT Labs
- Cybercrime acceleration slows, but still grows

The Catalyst

- Education increasingly re-adopts FM
- Governments mandate FM for critical software
- Rowan Carter: automated loop verification
- Breakthrough reduces manual reasoning burden

The End

- CI pipelines verify complex control flow automatically
- Critical industries reach near-universal coverage
- Cybercrime mitigated by verified software
- FM becomes invisible infrastructure

Contact

https://charles.systems/

charles@utdallas.edu

Cook, B.
[PLMW@PLDI24] Thoughts on the interplay between corporate, government, and university R&D, Jul 2024.