Let’s play...

Devil’s Advocate

I will give you one of Brooks’s claims and a rebuttal.

How would Brooks argue against the rebuttal?

Is he right?

Brooks makes a lot of big claims, sometimes with no evidence. But is he right?
**Claim:** Software entities are more complex for their size than perhaps any other human construct because no two parts are alike (at least above the statement level).

**Rebuttal:** Software has common parts at the domain level (e.g., accounting packages) and the programming level (e.g., data structures).

Brooks on data structures: Yes, but that is slowing down — and it’s at the code level, really.
Brooks on application structures: Software operates in complex environments, created by humans, and these environments change. So do users’ expectations.
Claim: The software entity is constantly subject to pressures for change. Of course, so are buildings, cars, computers. But manufactured things are infrequently changed after manufacture...

Rebuttal: Software companies do this, too. Microsoft Windows and Mac OS X are released as “new models”. Most software packages are.

Brooks: Shrink-wrap software is but a small portion of software in the world. Most software is built in-house or on-spec for custom applications. People’s needs change and grow over time, and the cost of change is much less.
complexity

Essence 1 of software.
software system >> digital computer >> most things people build — # of distinct states
conformity

Essence 2 of software. Software must conform to human-designed interfaces. Much worse than physics!? Human complexity is *arbitrary* and *particular*.
Changeability

Essence 3 of software.
The cost of change is much less than in traditional, material design.
invisibility

Essence 4 of software.
People don’t see it, so think it is easy to change.
    ... but: “The computer won’t let us.”
Software is not constrained in space.  We can model it in an arbitrary number of ways.
Still: We have components with data flows between them.
Fred Brooks

IBM System 360

OS/360

This guy knows what he is talking about.
Don’t bow to authority, but respect experience and understanding.
“Past breakthroughs solved accidental difficulties.”
... difficulties in expressing solutions. These are at the programming level.

Timesharing? /remember the past.../
Unified environments? IDEs: Eclipse, NetBeans, ... Dr. Scheme, Dr. Java, JES, GNAT
Ada and other high-level languages
object-oriented programming
artificial intelligence
knowledge-based systems
automatic programming
graphical programming
program verification
better tools and computers

“Hopes for the silver” — approaches that have failed or will fail.
... difficulties in expressing solutions. These are at the programming level.

My career: AI, KBS, OOP, HLL moving targets, seamless modeling, generate what we can
Most are incremental, not orders–of–magnitude. We run into scale and ill–defined problems.
“Promising attacks on the ... essence” — approaches that offer hope of incremental advance. Buying works (only) for stock problems. But everyone wants to tinker. See: Collab Suite. The other two: “Grow software, don’t build it.” Yes! The agile approaches. But not silver.
Final “promising attack”

Yes!

But you cannot mass-produce Mozart, or Steve Jobs, or Fred Brooks. (a composer/artist?)
What could a university CS program do to create great designers—or at least better designers?

My thoughts:
Build more systems. Build bigger systems. Get feedback from other designers. Work with users.
- open source projects
- “studio courses” a la architecture
Students: practice, practice, practice!