

Software Engineering

Architecture, Design and Patterns

Wladimir Cardoso Brandão
www.wladimirbrandao.com



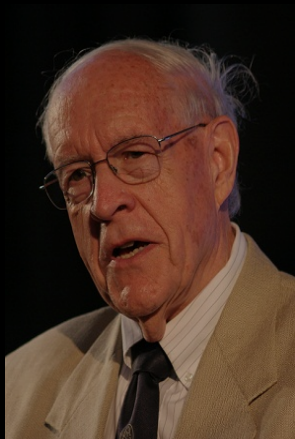
Department of Computer Science (DCC)
Institute of Exact Sciences and Informatics (ICEI)
Pontifical Catholic University of Minas Gerais (PUC Minas)

The author thanks the support to:
Students who willingly help me in this task



CHAPTER 02

NO SILVER BULLET



There is no single development,
in either technology or management technique, which by
itself promises even one order-
of-magnitude improvement
within a decade in producti-
vity, in reliability, in simplicity

*Fred Brooks, 1986
Turing Award, 1999*



Brook's classic essay → "Why is SE so hard?"

"There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity"

Fred Brooks, 1986

- ▶ Silver Bullet → single approach that by itself can deliver one order-of-magnitude improvement to some aspect of software development
- ▶ One order-of-magnitude → 10 times



- ▶ Problems facing SE:
 - ▶ Essence → intrinsic difficulties (domain)
 - ▶ Accident → difficulties related to the production
- ▶ Most techniques attack the accidents
- ▶ For one order-of-magnitude improvement:
 - ▶ Accidents → should account for 90% of the overall effort
 - ▶ Tools → should reduce accidents to zero
- ▶ But in fact:
 - ▶ Accidents → do not account for 90% of the overall effort
 - ▶ New tool or technique solves some problems while introducing others



- ▶ Complexity → problems can be complex
- ▶ Conformity → requirements change
- ▶ Changeability → pressure to change
- ▶ Invisibility → Software is invisible and intangible, hard to get “big picture”



- ▶ Software entities are complex
 - ▶ Different from materials in other domains, no two entities are alike in software
- ▶ Huge number of states
 - ▶ Large software systems have an order of magnitude more states than computers (hardware) do
- ▶ Exponentially increasing
 - ▶ As the size of a system increases, the number, and particularly the types of parts increase exponentially



- ▶ It is impossible to abstract away the complexity of the application domain
 - ▶ For instance → air traffic control, international banking, avionics software
- ▶ Domains are intrinsically complex and this complexity will appear in the software as designers attempt to model the domain
- ▶ Complexity also comes from the numerous and tight relationships between heterogeneous software artifacts such as specs, docs, code, test cases, etc.



- ▶ Problems resulting from complexity:
 - ▶ difficult team communication
 - ▶ product flaws; cost overruns; schedule delays
 - ▶ personnel turnover (loss of knowledge)
 - ▶ unenumerated states (lots of them)
 - ▶ lack of extensibility (complexity of structure)
 - ▶ unanticipated states (security loopholes)
 - ▶ project overview is difficult



- ▶ Arbitrariness → unplanned changes
 - ▶ Client demands → dealing with a change in vendor imposed by the customer
 - ▶ Regulation issues → implementing rules that may change annually
 - ▶ Environment changes → adapting to changes or to a pre-existing environment, such as integrating with legacy systems
 - ▶ Other issues → a new CIO arrives at the company decided to “make a mark” by completely changing the business process
- ▶ It is almost impossible to plan for arbitrary change. We just have to wait and deal with it when it happens



- ▶ Pressure to change → culture, intangibility
- ▶ In tangible domains clients understand how difficult (and expensive) reworking is
 - ▶ Imagine asking for a new layout of a house after the foundation has been poured
- ▶ Manufactured products are rarely changed after they have been created
 - ▶ Automobiles are infrequently recalled
 - ▶ Buildings are expensive to remodel
- ▶ But software? Constantly asked to change
 - ▶ Clients often don't understand enough about software to understand when a change request requires significant rework



- ▶ Representation → intangibility
- ▶ Different from tangible products that:
 - ▶ Is easy to represent
 - ▶ Blueprints → geometry identifies problems
 - ▶ After manufactured rarely changed
 - ▶ Imagine asking for a new layout of a house after the foundation has been poured
 - ▶ Automobiles → infrequently recalled
 - ▶ Buildings → expensive to remodel
- ▶ Software is invisible and intangible:
 - ▶ Difficult to represent
 - ▶ Several notations and diagrams
 - ▶ Constantly asked to change
 - ▶ Clients often don't understand about SE to understand when a change request requires significant rework



- ▶ Difficult to design graphical displays of software that convey meaning to developers
- ▶ Difficult to reduce software to diagrams
 - ▶ UML → 13 diagram types to model class structure, object relationships, activities, event handling, software architecture, deployment, packages...
 - ▶ Notations of the different types almost never appear in the same diagram, really documenting 13 different aspects
- ▶ Hard to get both “the big picture” and detailed view at the same time
- ▶ Lack of visualization deprives engineers from using the brain’s powerful visual skills







- ▶ Buy x Build → don't develop software when you can avoid it
- ▶ Rapid prototyping → clarify requirements
- ▶ Incremental development → don't build software, grow it
- ▶ Great designers → be on the look out for them, when you find them, don't let go

No single technique produces an order of magnitude increase by itself, but several techniques together can achieve it, requiring industry-wide enforcement and discipline



- ▶ Programming environments
- ▶ Programming languages
- ▶ Time sharing
- ▶ OO analysis and design



-  Sommerville, Ian
Software Engineering.
10ed, 2016.
-  Larman, Graig
Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.
3ed, 2004.
-  Pressman, Roger; Maxim, Bruce
Software Engineering: A Practitioner's Approach.
8ed, 2014.
-  Booch, Grady; Rumbaugh, James; Jacobson, Ivar
The Unified Modeling Language User Guide.
2ed, 2005.



Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John

Design Patterns: Elements of Reusable Object-Oriented Software.

1ed, 1994.



Buschmann, Frank; Rohnert, Hans; Stal, Michael; Sommerlad, Peter; Meunier, Regine

Pattern-Oriented Software Architecture, A System of Patterns.

1ed, 1996.

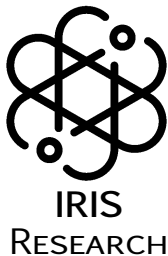


Frederick P. Brooks

The Mythical Man-Month.

Proceedings of the IFIP Tenth World Computing Conference, 1986.

THANK YOU



Wladimir Cardoso Brandão

www.wladimirbrandao.com

wladimir@pucminas.br



"Science is more than a body of knowledge. It is a way of thinking."

Carl Sagan