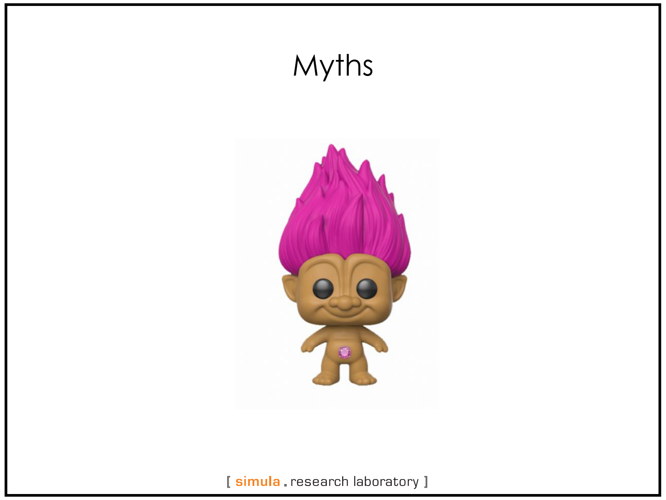
Magne Jørgensen,
SimulaMet, Scienta and Kathmandu University

From myths and fashions to evidence-based software engineering

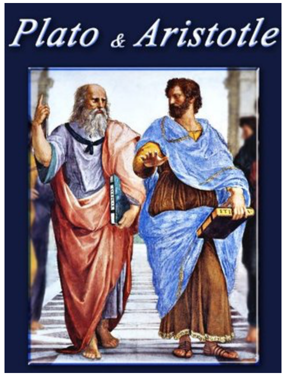


Myths

[simula . research laboratory]

Fashion

Evidence-based judgments and decisions

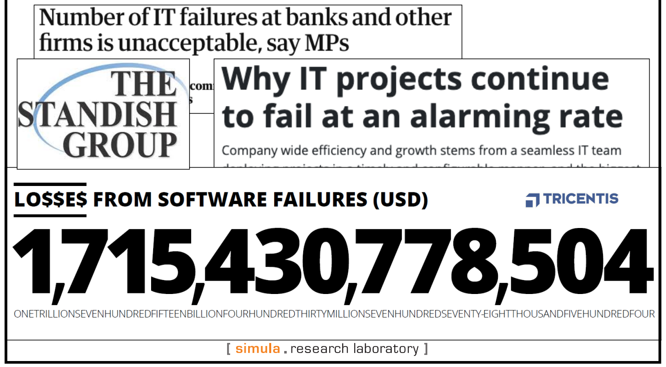


LET'S START WITH SOME MYTHS AND WHY WE BELIEVE IN THEM

The paper clip was invented by a Norwegian

Most communication is non-verbal

Software projects fail most of the time, and the quality of software is low



Number of IT failures at banks and other firms is unacceptable, say MPs

THE STANDISH GROUP

com

Why IT projects continue to fail at an alarming rate

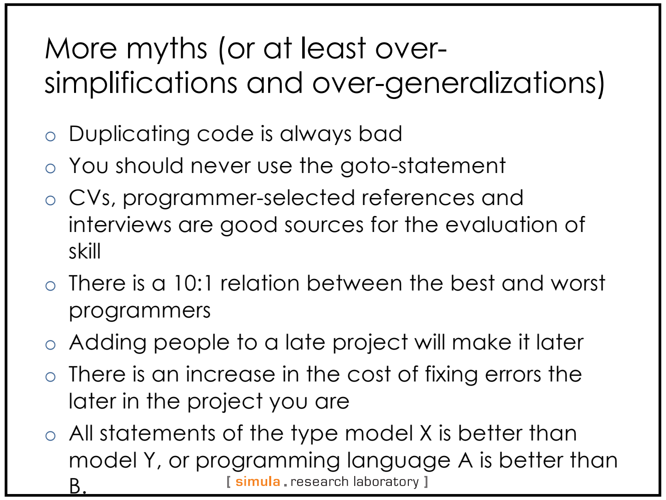
Company wide efficiency and growth stems from a seamless IT team

LO$$E$ FROM SOFTWARE FAILURES (USD)

TRICENTIS

1,715,430,778,504

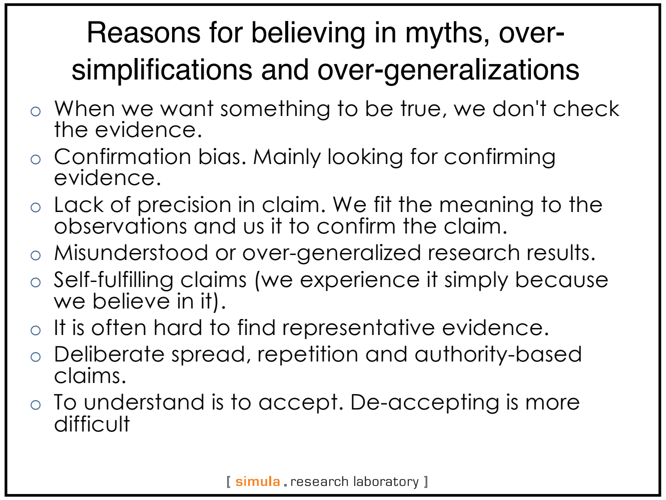
ONETRILLIONSEVENHUNDREDFIFTEENBILLIONFOURHUNDREDTHIRTYMILLIONSEVENHUNDREDSEVENTY-EIGHTTHOUSANDFIVEHUNDREDFOUR



More myths (or at least over-simplifications and over-generalizations)

- Duplicating code is always bad
- You should never use the goto-statement
- CVs, programmer-selected references and interviews are good sources for the evaluation of skill
- There is a 10:1 relation between the best and worst programmers
- Adding people to a late project will make it later
- There is an increase in the cost of fixing errors the later in the project you are
- All statements of the type model X is better than model Y, or programming language A is better than B.

[simula . research laboratory]



Reasons for believing in myths, over-simplifications and over-generalizations

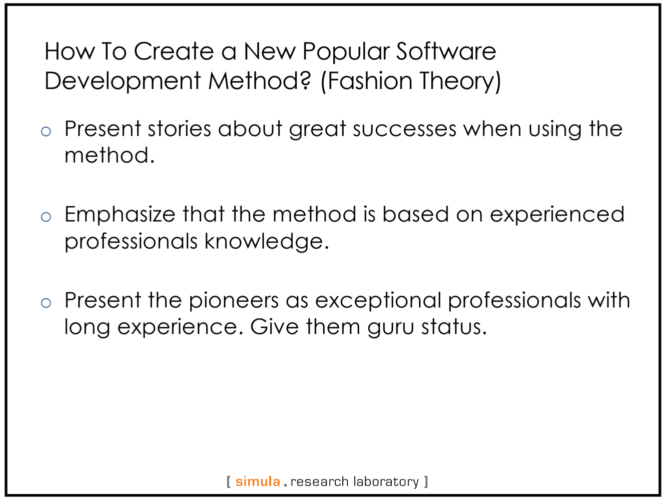
- When we want something to be true, we don't check the evidence.
- Confirmation bias. Mainly looking for confirming evidence.
- Lack of precision in claim. We fit the meaning to the observations and us it to confirm the claim.
- Misunderstood or over-generalized research results.
- Self-fulfilling claims (we experience it simply because we believe in it).
- It is often hard to find representative evidence.
- Deliberate spread, repetition and authority-based claims.
- To understand is to accept. De-accepting is more difficult

[simula . research laboratory]

A FEW WORDS ON FASHION

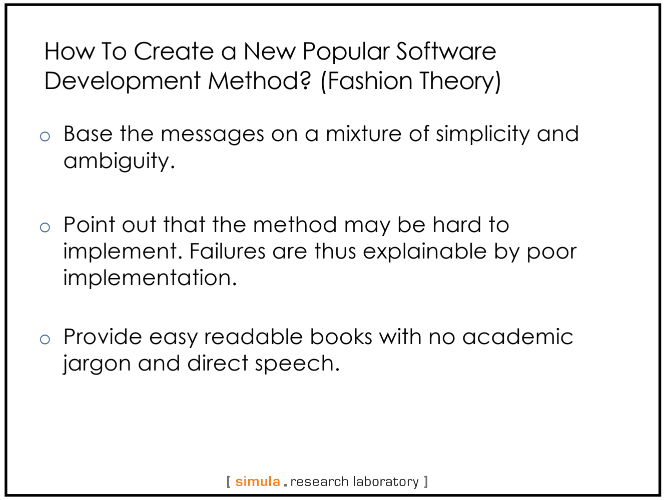
How To Create a New Popular Software Development Method? (according to the Fashion Theory – Kieser)

1. Present one key principle that, according to the gurus, has been neglected in previous methods.
2. Describe how the old methods are bound to fail.
3. Link the new method with highly treasured values, such as communication, individuals, flexibility and user value.



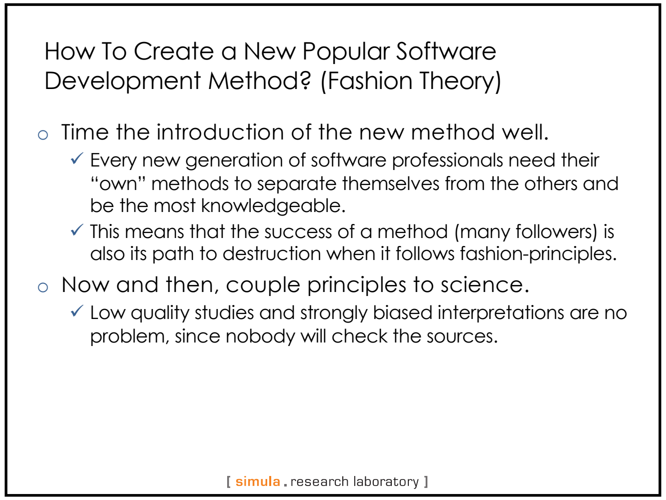
How To Create a New Popular Software Development Method? (Fashion Theory)

- Present stories about great successes when using the method.
- Emphasize that the method is based on experienced professionals knowledge.
- Present the pioneers as exceptional professionals with long experience. Give them guru status.



How To Create a New Popular Software Development Method? (Fashion Theory)

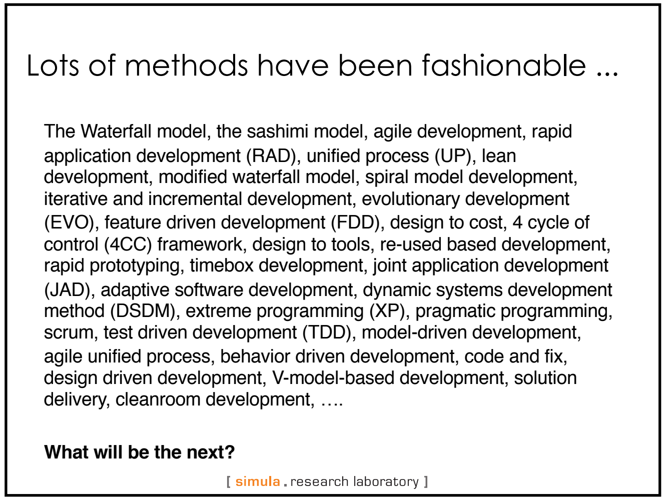
- Base the messages on a mixture of simplicity and ambiguity.
- Point out that the method may be hard to implement. Failures are thus explainable by poor implementation.
- Provide easy readable books with no academic jargon and direct speech.



How To Create a New Popular Software Development Method? (Fashion Theory)

- Time the introduction of the new method well.
  - ✓ Every new generation of software professionals need their "own" methods to separate themselves from the others and be the most knowledgeable.
  - ✓ This means that the success of a method (many followers) is also its path to destruction when it follows fashion-principles.
- Now and then, couple principles to science.
  - ✓ Low quality studies and strongly biased interpretations are no problem, since nobody will check the sources.

[simula . research laboratory]



Lots of methods have been fashionable ...

The Waterfall model, the sashimi model, agile development, rapid application development (RAD), unified process (UP), lean development, modified waterfall model, spiral model development, iterative and incremental development, evolutionary development (EVO), feature driven development (FDD), design to cost, 4 cycle of control (4CC) framework, design to tools, re-used based development, rapid prototyping, timebox development, joint application development (JAD), adaptive software development, dynamic systems development method (DSDM), extreme programming (XP), pragmatic programming, scrum, test driven development (TDD), model-driven development, agile unified process, behavior driven development, code and fix, design driven development, V-model-based development, solution delivery, cleanroom development, ….

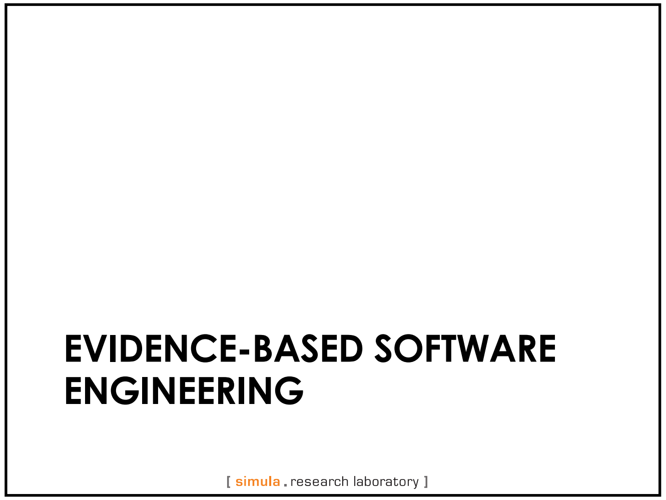
What will be the next?

[simula . research laboratory]

FASHION CAN BE GOOD ...

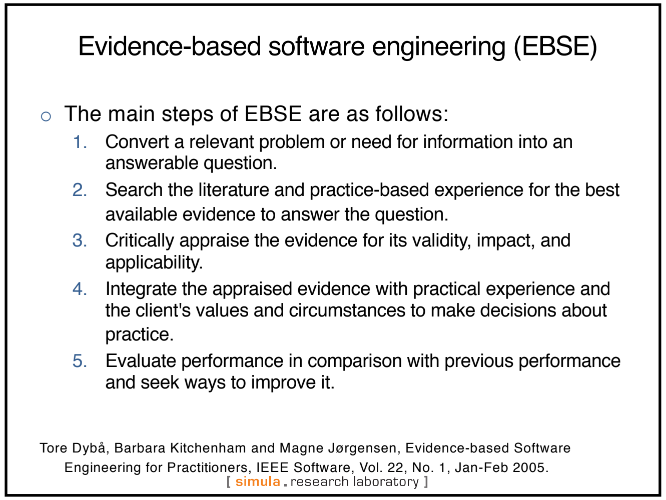
BUT WOULDN'T IT BE BETTER IF THE CHANGES WERE EVIDENCE-BASED?

[simula . research laboratory]



EVIDENCE-BASED SOFTWARE ENGINEERING

[simula . research laboratory]

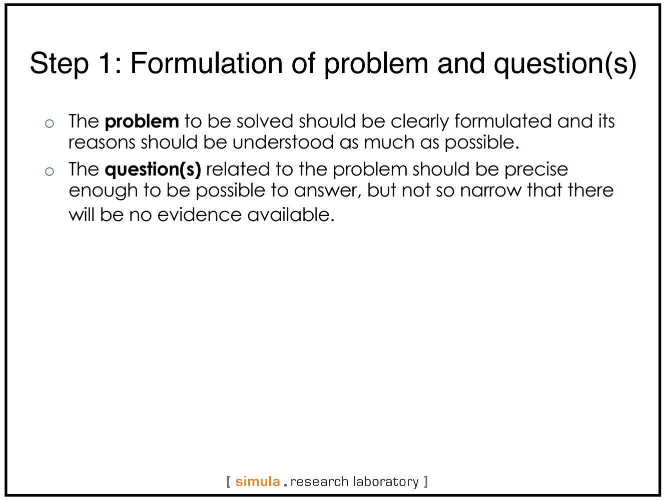


Evidence-based software engineering (EBSE)

- The main steps of EBSE are as follows:
  1. Convert a relevant problem or need for information into an answerable question.
  2. Search the literature and practice-based experience for the best available evidence to answer the question.
  3. Critically appraise the evidence for its validity, impact, and applicability.
  4. Integrate the appraised evidence with practical experience and the client's values and circumstances to make decisions about practice.
  5. Evaluate performance in comparison with previous performance and seek ways to improve it.

Tore Dybå, Barbara Kitchenham and Magne Jørgensen, Evidence-based Software Engineering for Practitioners, IEEE Software, Vol. 22, No. 1, Jan-Feb 2005.

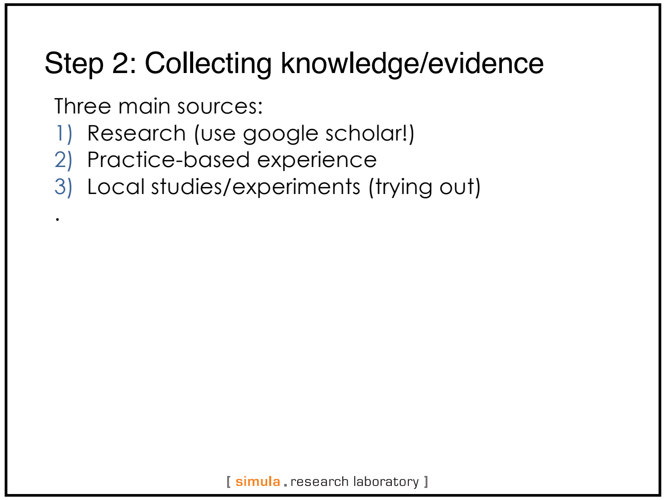
[simula . research laboratory]



Step 1: Formulation of problem and question(s)

- The **problem** to be solved should be clearly formulated and its reasons should be understood as much as possible.
- The **question(s)** related to the problem should be precise enough to be possible to answer, but not so narrow that there will be no evidence available.

[simula . research laboratory]



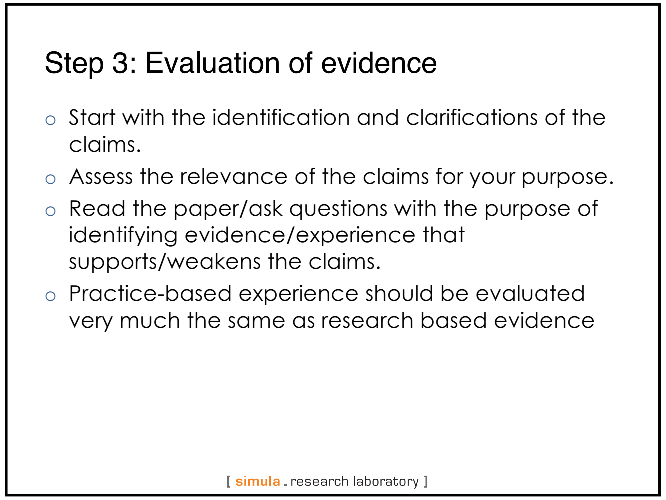
Step 2: Collecting knowledge/evidence

Three main sources:

1) Research (use google scholar!)
2) Practice-based experience
3) Local studies/experiments (trying out)

.

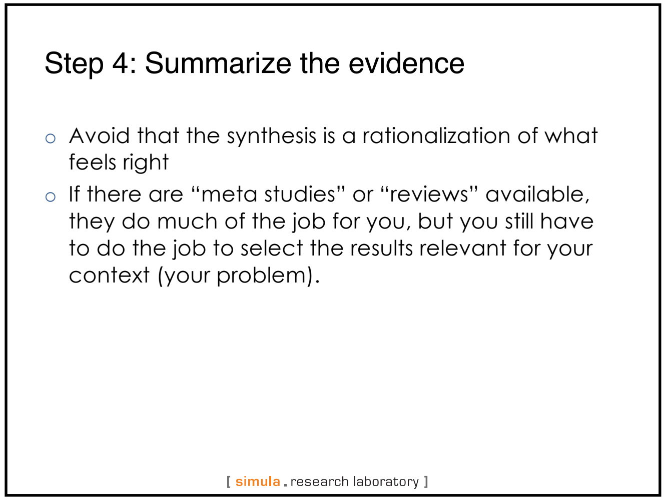
[simula . research laboratory]



Step 3: Evaluation of evidence

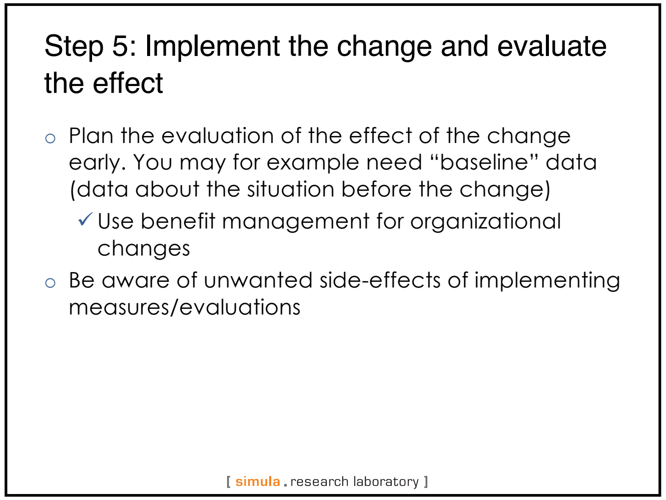
- Start with the identification and clarifications of the claims.
- Assess the relevance of the claims for your purpose.
- Read the paper/ask questions with the purpose of identifying evidence/experience that supports/weakens the claims.
- Practice-based experience should be evaluated very much the same as research based evidence

[simula . research laboratory]



Step 4: Summarize the evidence

- Avoid that the synthesis is a rationalization of what feels right
- If there are “meta studies” or “reviews” available, they do much of the job for you, but you still have to do the job to select the results relevant for your context (your problem).



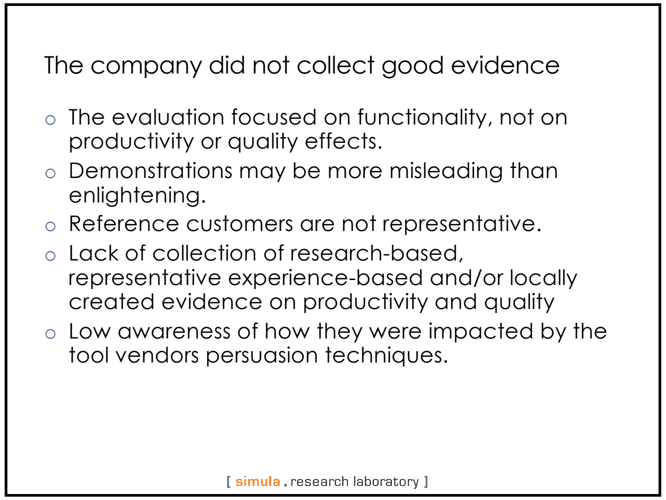
Step 5: Implement the change and evaluate the effect

- Plan the evaluation of the effect of the change early. You may for example need “baseline” data (data about the situation before the change)
  - ✓ Use benefit management for organizational changes
- Be aware of unwanted side-effects of implementing measures/evaluations

A REAL-LIFE STORY

Changing the development tool in a company

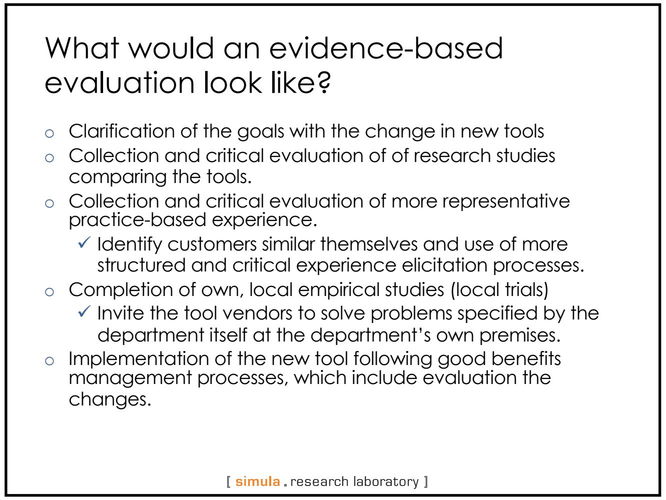
- A software development department wanted to replace their “old-fashioned” development tool with a more modern and more efficient one.
- They visited many possible vendors, participated at numerous demonstrations, and contacted several “reference customers”. Finally, they chose a development tool.
- A couple of years after the change, the department measured the change in development efficiency. Unfortunately, this had not improved and the new development tool was not as good as expected.
- What went wrong?



The company did not collect good evidence

- The evaluation focused on functionality, not on productivity or quality effects.
- Demonstrations may be more misleading than enlightening.
- Reference customers are not representative.
- Lack of collection of research-based, representative experience-based and/or locally created evidence on productivity and quality
- Low awareness of how they were impacted by the tool vendors persuasion techniques.

[simula . research laboratory]

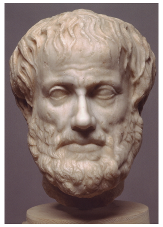


What would an evidence-based evaluation look like?

- Clarification of the goals with the change in new tools
- Collection and critical evaluation of of research studies comparing the tools.
- Collection and critical evaluation of more representative practice-based experience.
  - ✓ Identify customers similar themselves and use of more structured and critical experience elicitation processes.
- Completion of own, local empirical studies (local trials)
  - ✓ Invite the tool vendors to solve problems specified by the department itself at the department's own premises.
- Implementation of the new tool following good benefits management processes, which include evaluation the changes.

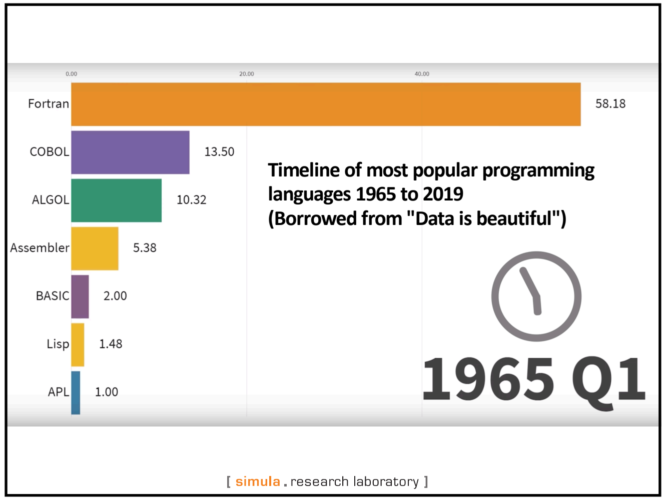
[simula . research laboratory]

Follow in the footsteps of Aristotles and base practices on good evidence!



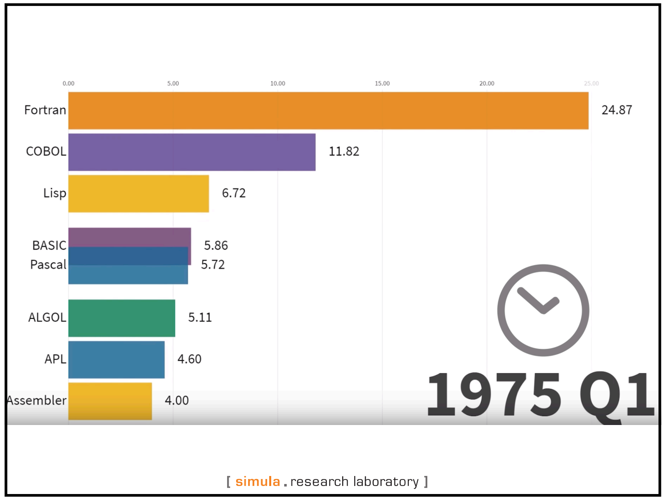
PROBLEM
QUESTION
SEARCH
EVALUATION
AGGREGATION
IMPLEMENTATION
EVALUATION

QUESTIONS?

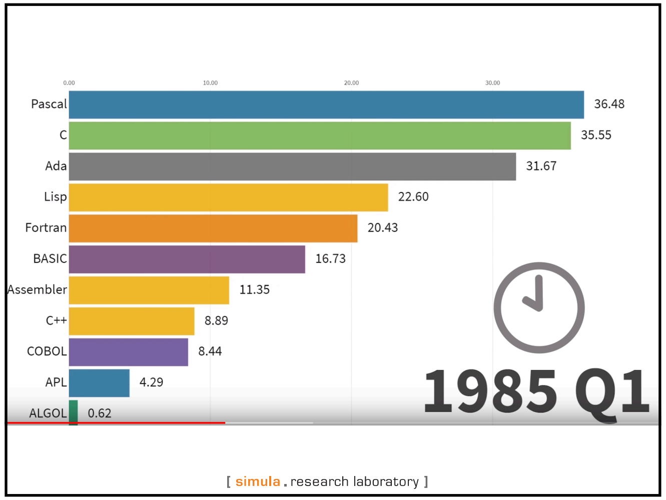


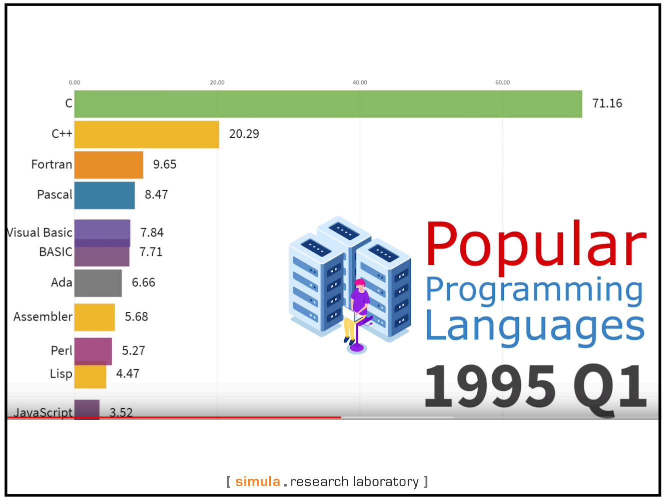
Timeline of most popular programming languages 1965 to 2019 (Borrowed from "Data is beautiful")

[simula . research laboratory]

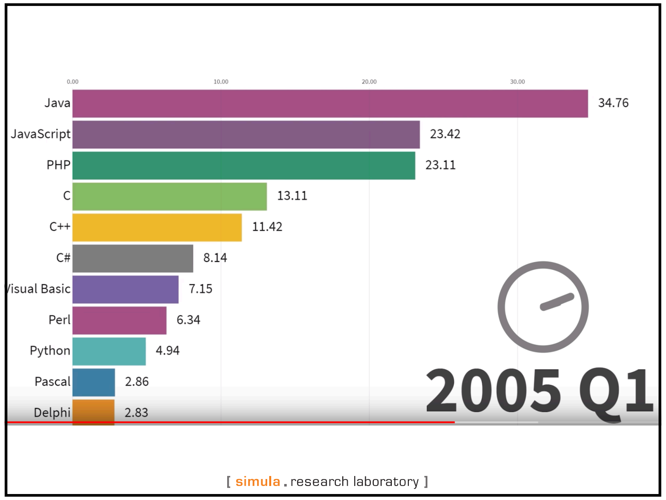


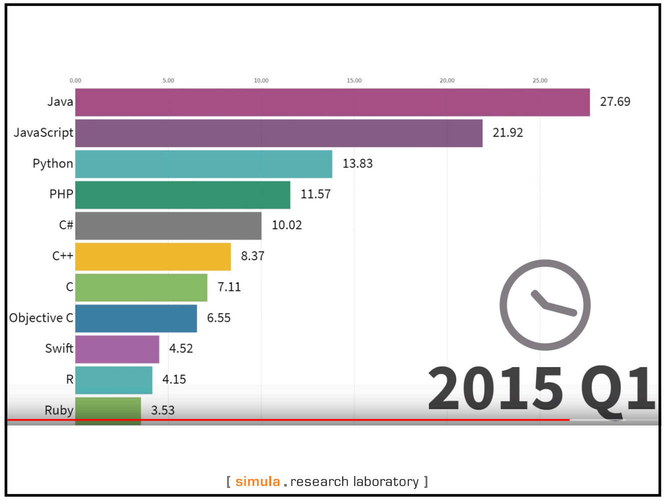
[simula . research laboratory]

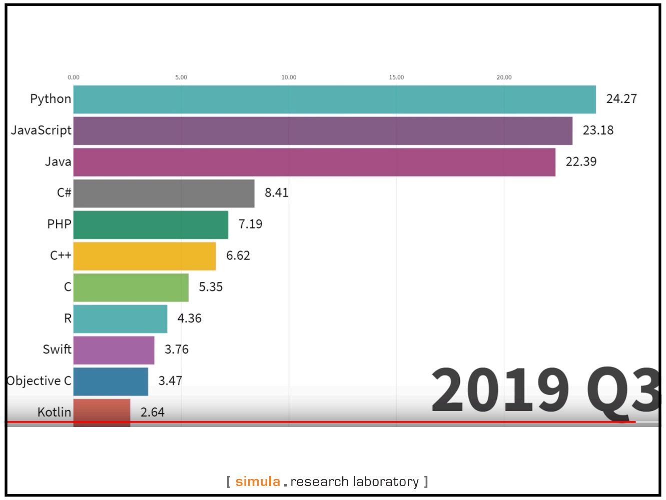

0.00
10.00
20.00
30.00
Pascal 36.48
C 35.55
Ada 31.67
Lisp 22.60
Fortran 20.43
BASIC 16.73
Assembler 11.35
C++ 8.89
COBOL 8.44
APL 4.29
ALGOL 0.62
1985 Q1
[simula . research laboratory]


0.00
20.00
40.00
60.00
C 71.16
C++ 20.29
Fortran 9.65
Pascal 8.47
Visual Basic 7.84
BASIC 7.71
Ada 6.66
Assembler 5.68
Perl 5.27
Lisp 4.47
JavaScript 3.52
Popular
Programming
Languages
1995 Q1
[simula . research laboratory]

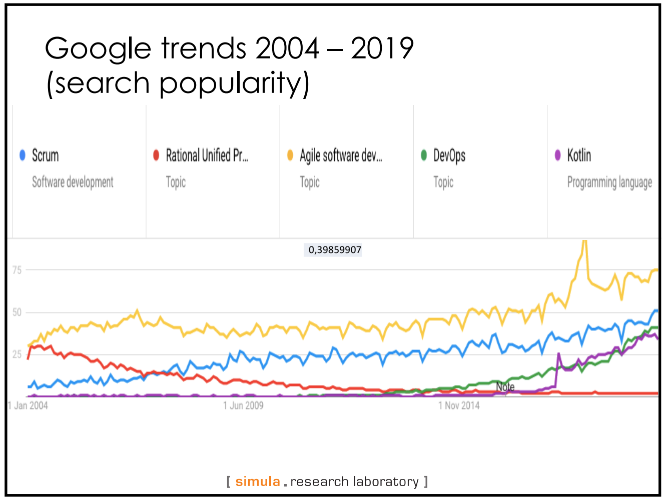
11/14/19


0.00
10.00
20.00
30.00
Java 34.76
JavaScript 23.42
PHP 23.11
C 13.11
C++ 11.42
C# 8.14
isual Basic 7.15
Perl 6.34
Python 4.94
Pascal 2.86
Delphi 2.83
2005 Q1
[simula . research laboratory]


0.00
5.00
10.00
15.00
20.00
25.00
Java 27.69
JavaScript 21.92
Python 13.83
PHP 11.57
C# 10.02
C++ 8.37
C 7.11
Objective C 6.55
Swift 4.52
R 4.15
Ruby 3.53
2015 Q1
[simula . research laboratory]



[simula . research laboratory]



Google trends 2004 – 2019
(search popularity)

[simula . research laboratory]

The ease of affecting beliefs: Are risk-willing or risk-averse developers better?

Group

Initially
Average 3.3
Debriefing
Average 2: 3.5
2 weeks later
Average 3:
3.5

Group
–
Initially
Average 5.4
Debriefing
Average 2: 5.0
2 weeks later
Average 3:
4.9

Study design: Resea✗h evidence + Self-generated argument.

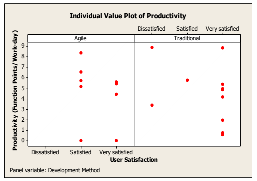
Question: Based on your experience, do you think that risk-willing programmers are better than risk-averse programmers?

1 (totally agree) – 5 (No difference) - 10 (totally disagree)

Neutral group: Average 5.0

"I see it when I believe it" vs "I believe it when I see it"

- **Design:**
  - ✓ Data sets with randomly set performance data comparing "traditional" and "agile" methods.
  - ✓ Survey of each developer's belief in agile methods
- **Question**: How much do you, based on the data set, agree in: *"Use of agile methods has caused a better performance when looking at the combination of productivity and user satisfaction."*
- **Result**:
  - ✓ Previous belief in agile determined what they saw in the random data


Individual Value Plot of Productivity
Agile
Traditional
Dissatisfied
Satisfied
Very satisfied
Productivity (Function Points/Work-day)
User Satisfaction
Panel variable: Development Method