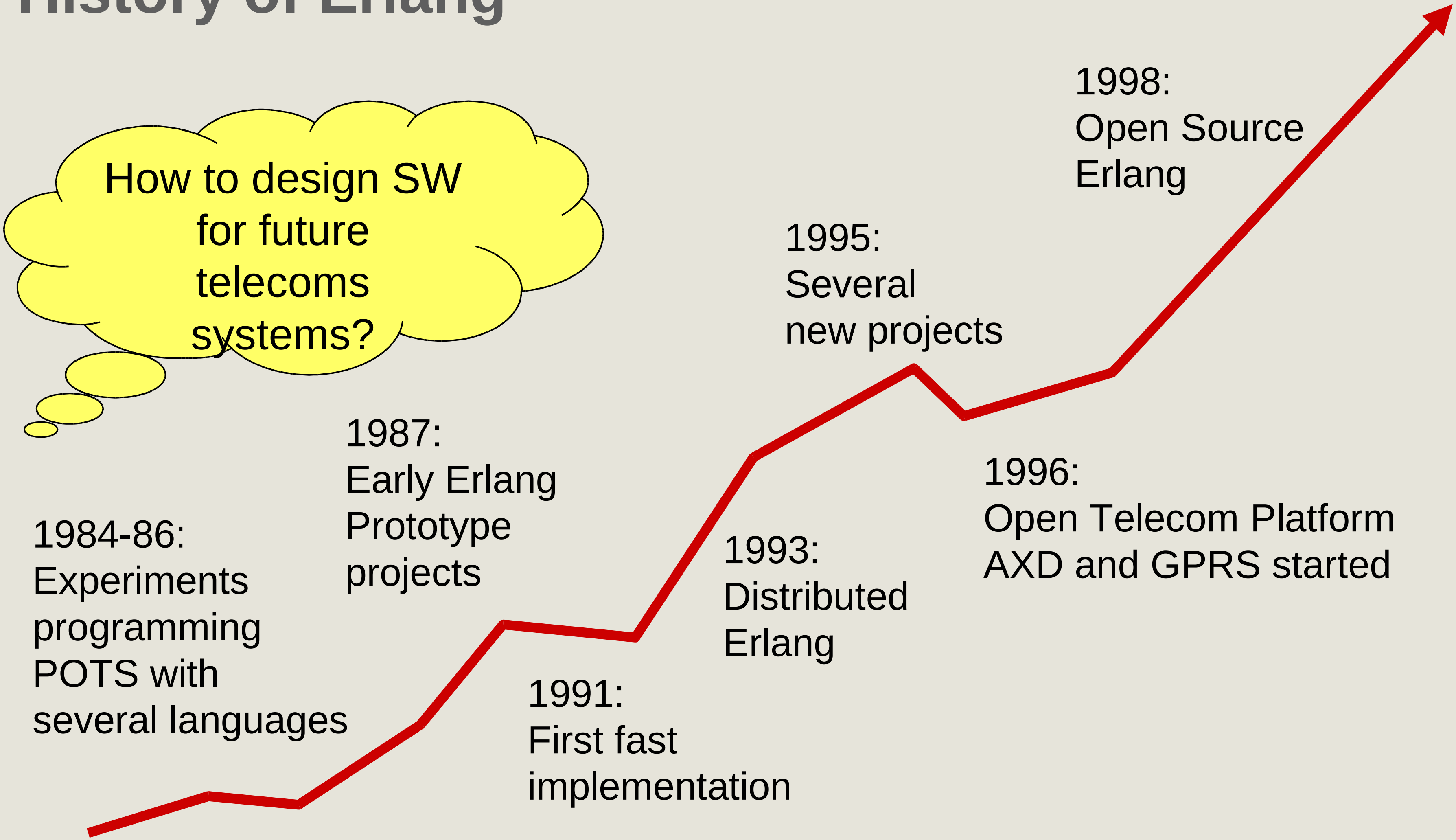
20 Years of Commercial Functional Programming

Ulf Wiger
Senior Software Architect
Ericsson AB

History of Erlang



How to design SW for future telecoms systems?

1984-86:
Experiments
programming
POTS with
several languages

1987:
Early Erlang
Prototype
projects

1991:
First fast
implementation

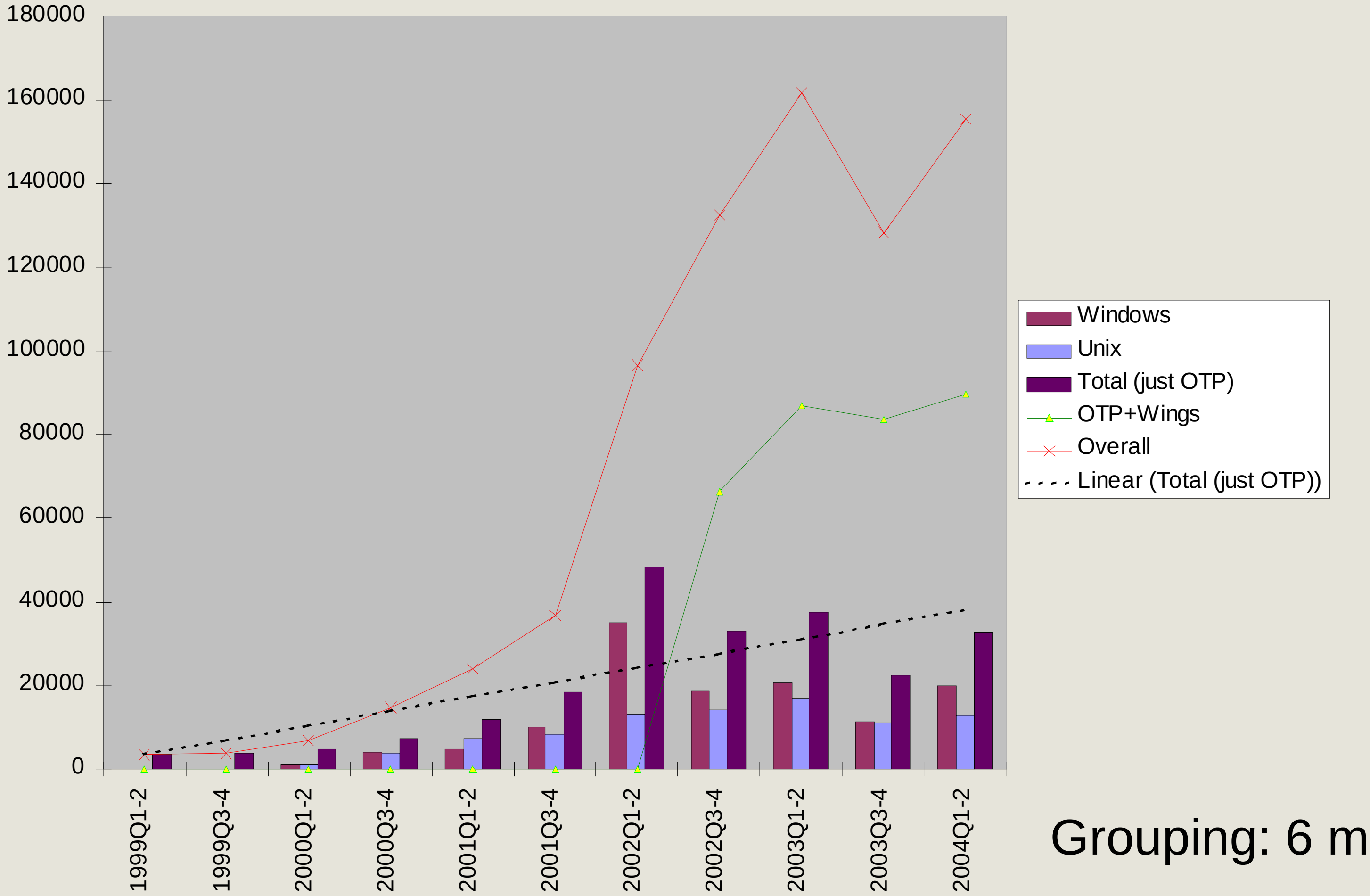
1993:
Distributed
Erlang

1995:
Several
new projects

1996:
Open Telecom Platform
AXD and GPRS started

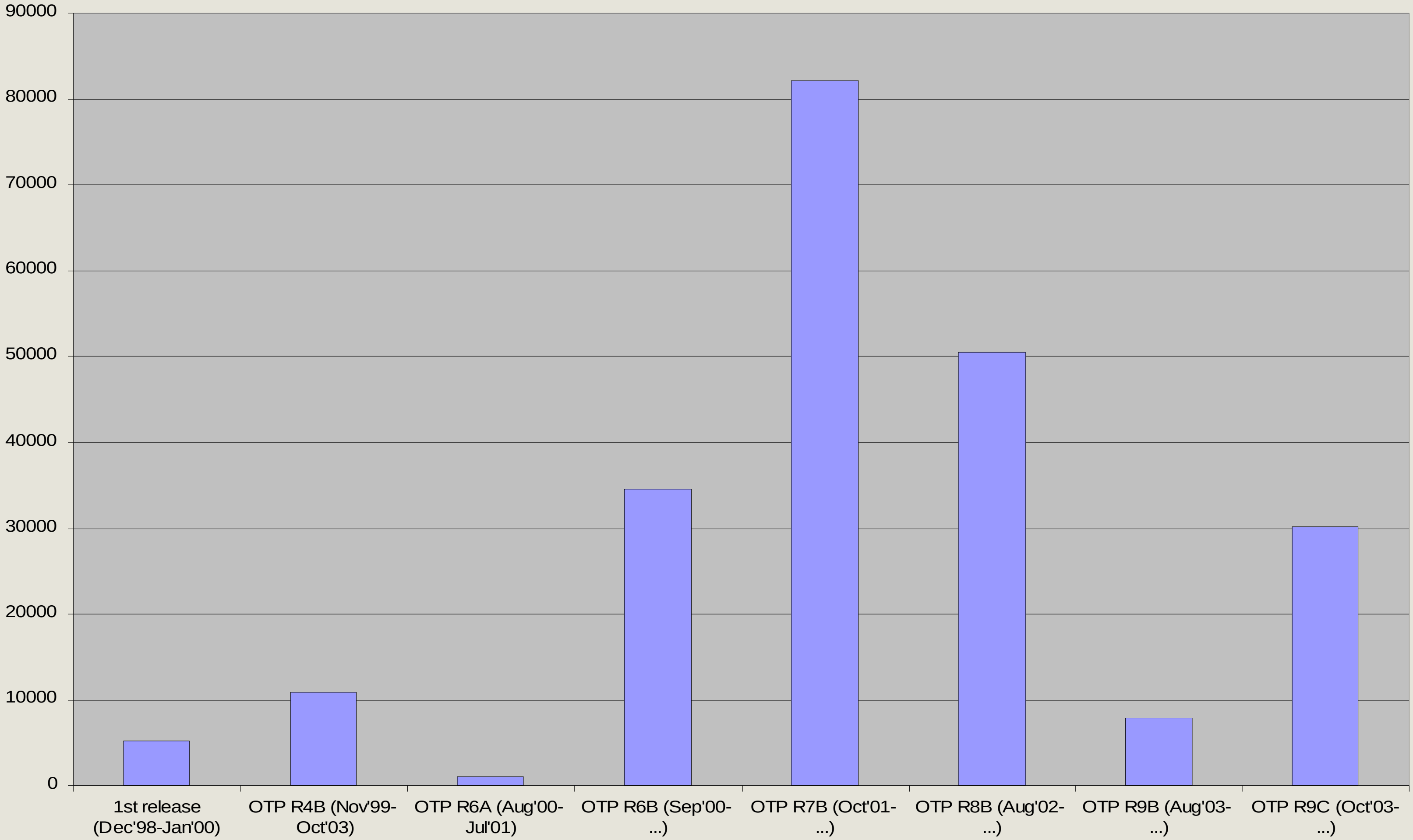
1998:
Open Source
Erlang

Downloads since Open Source Launch ’98



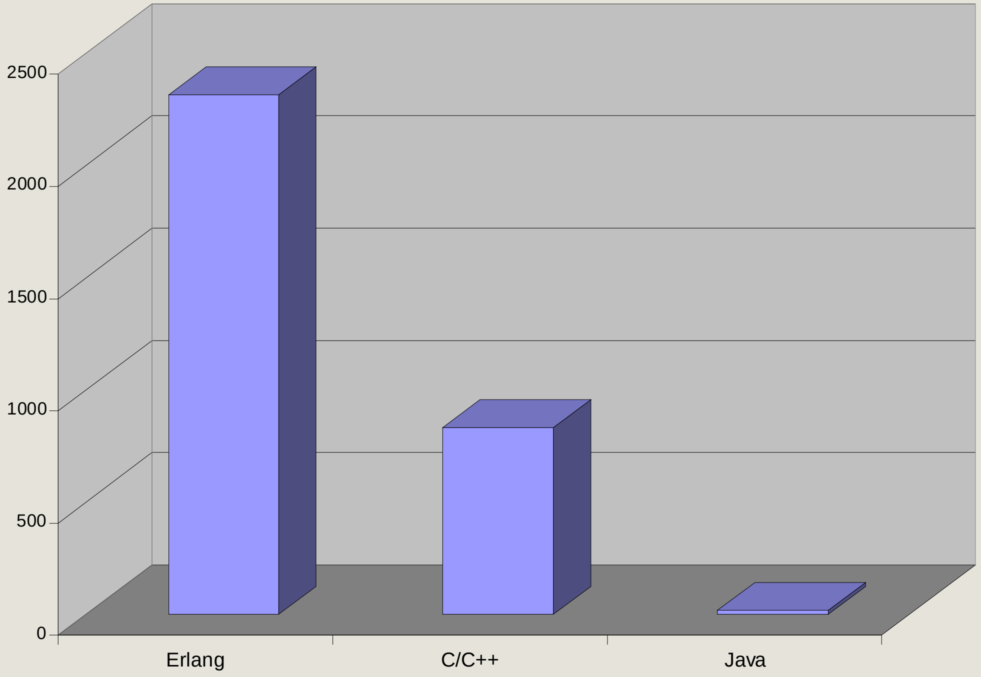
Grouping: 6 months

Total Downloads per Erlang/OTP Release



Erlang is a Systems Programming Language

- Erlang is ideal for:
  - Complex control logic
  - Concurrency/state machines
  - Program supervision
  - Distribution/redundancy
- The whole system comprises:
  - Hardware
  - Hardware control (drivers)
  - Bootstrap logic (C, shell scripts)
  - Application logic
  - Often, 3rd party software
  - Operator GUI (HTML, Java, ...)


KLOCs, AXD 301 Control System

The challenge of technology introduction

- Most difficulties facing technology introduction have little to do with the actual technology.
- The perfect objection to new technology is one that:
  - Sounds quite technical and well considered.
  - Is difficult/impossible to prove or disprove.
- Instead of looking past the actual objection, engineers will spin their wheels trying to refute it.
  - ... and even if they succeed, few people will care.
- The only things that might save you in the end, are commercial success, predictability and high quality.

Case in point: Erlang

- Everyone in Industry ”knows” that:
  - Erlang isn’t fast enough.
  - Erlang isn’t OO (well, this is fortunately true!)
  - It would be too hard to recruit Erlang programmers.
  - If Erlang is indeed better at something,
    the Mainstream will catch up soon enough
    (always in the Next Release™.)
  - If an Erlang project succeeds,
    it’s due to excellent project management
    or unusually skilled designers.
  - If an Erlang project fails, it’s proof that Erlang isn’t better.

Objection: Erlang isn’t fast enough

- No competing product outperforms Ericsson’s ENGINE solution for Telephony over packet-based networks.
- No competing product outperforms Ericsson’s Erlang-based GPRS Signalling Support Node (SGSN).
- No competing product outperforms Nortel’s Erlang-based SSL Offload Accelerator.

- Lesson learned: it’s nearly impossible to predict system performance based on low-level ”micro” benchmarks.
- Failure to manage complexity often kills performance.

Objection: It’s too hard to hire Erlang programmers

- Lesson learned: Well, you need to work at it, but not at all impossible.
- Difficult to hire expert programmers in general.
- Hiring expert C++ (or Java) programmers has proven quite difficult (due to their exceptionally high market value.)
- Establishing relationships with local universities (even in the U.S.) has proven fruitful for Erlang-based projects.
- Companies should not try to recruit programmers who know/accept only one language – look for solid Comp. Sci. competence instead. Erlang itself is not very difficult.

Objection: Mainstream will catch up

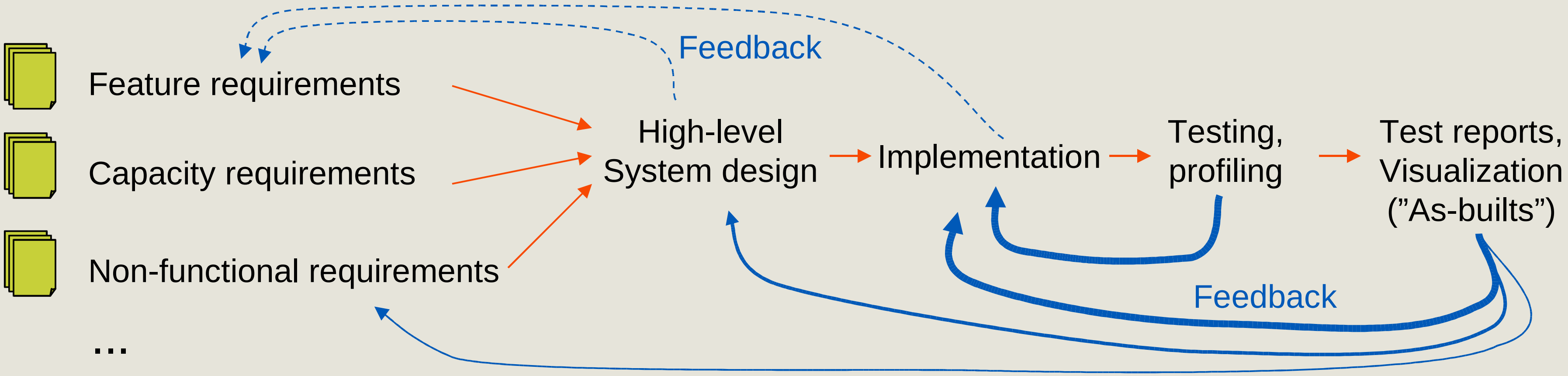
- 12 years...
- Still no serious contender to Erlang in the area of concurrency-oriented programming.
- Still waiting for the next Java compiler that will solve all problems.
- Waiting for UML 2.0 to address issues with concurrency & exception handling (then 3.0 for Executable UML, etc.)
- And Erlang is improving at a good pace.

Research Challenge:
How to determine a component’s ”footprint”?

- When a component has changed – how much do you need to re-test?
  - The programmer: “if it compiles, it works!”
  - The project manager: “we have to re-test everything!”
  - The truth normally lies somewhere in between, but where exactly?
- Testing is inevitable in industry, and <u>very</u> expensive.
- What parameters can affect testing outcome?
  - API changes (of course)
  - Message passing sequence & timing changes
  - Algorithmic complexity, timing, CPU & memory consumption
  - ...

Research Challenge: Specifications, implementation, and ”as-builts”

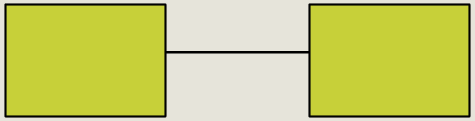
- SW design involves several orthogonal activities/teams
- How to exchange relevant info & close feedback loops?
- (Suggestion: Expressive programming languages coupled with different visualization tools and a non-formal to semi-formal requirements notation.)



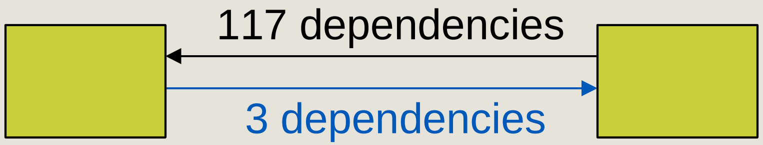
Examples of Visualization

- Jan Nyström, PhD, used static analysis to draw the (static) process tree of Erlang applications.
- Thomas Arts, PhD, hooked into Erlang’s trace support and generated state transition diagrams + exported trace analyses to model checking tools.
- Would like to see weighted block dependency diagrams generated through static analysis of code.

(UML) ...says little



...says more



117 dependencies

3 dependencies

When will we assist research projects?

(”we” = a large industrial development project)

- Continuously, on a small scale, e.g. with code samples, information, suggestions, ... (pro-bono)
- Now and then, we host a researcher and assist in running a small prototype – e.g. the HiPE team’s new type analyser. (low-risk, fun)
- Willing to support a small project if it’s likely to produce tangible results (=cost savings or improved product) within, say, 2 years. (exciting potential)