The Software Industry and Software Engineering

Dror Feitelson

Basic Seminar on Software Engineering
Hebrew University
2011

What Software Did You Use Today?

- We are usually unaware of most software...

What Software Companies Do You Know?

- Microsoft is not the only one...

Classification I

- Writing new software
  - OpenOffice / Firefox / Linux drivers
  - Computer games
  - Oracle database / ERP (enterprise resource planning)
- Integration of existing products
  - Computerize a garage / law office / warehouse
- We will focus on software production, not integration

Classification II

- Software contractors
  - Special software for specific use
  - Custom made for single client
- Corporate software products
  - Generic software for business use
  - Thousands of clients
- Mass market
  - Software for end-users (office / home)
  - Millions of clients

Classification III

- Program for self use
- Single programmer or small team, small project
- Medium size software project (20-30 people)
- Large software project (hundred+ people)

- We will focus on medium to large projects

Mary Shaw, “Prospects for an engineering discipline of software”. *IEEE Software* **7**(6), pp. 15-24, Nov.-Dec. 1990

CS professor at CMU since 1972
Chief scientist of SEI 1984-7
Co-director Sloan Software Industry Ctr. 2001-6
Fellow of the ACM, IEEE, AAAS

Software Engineering

- A label applied to a set of current practices for software development
- Not really an engineering discipline
- But has a potential to become one
- Insights by comparing with other engineering disciplines

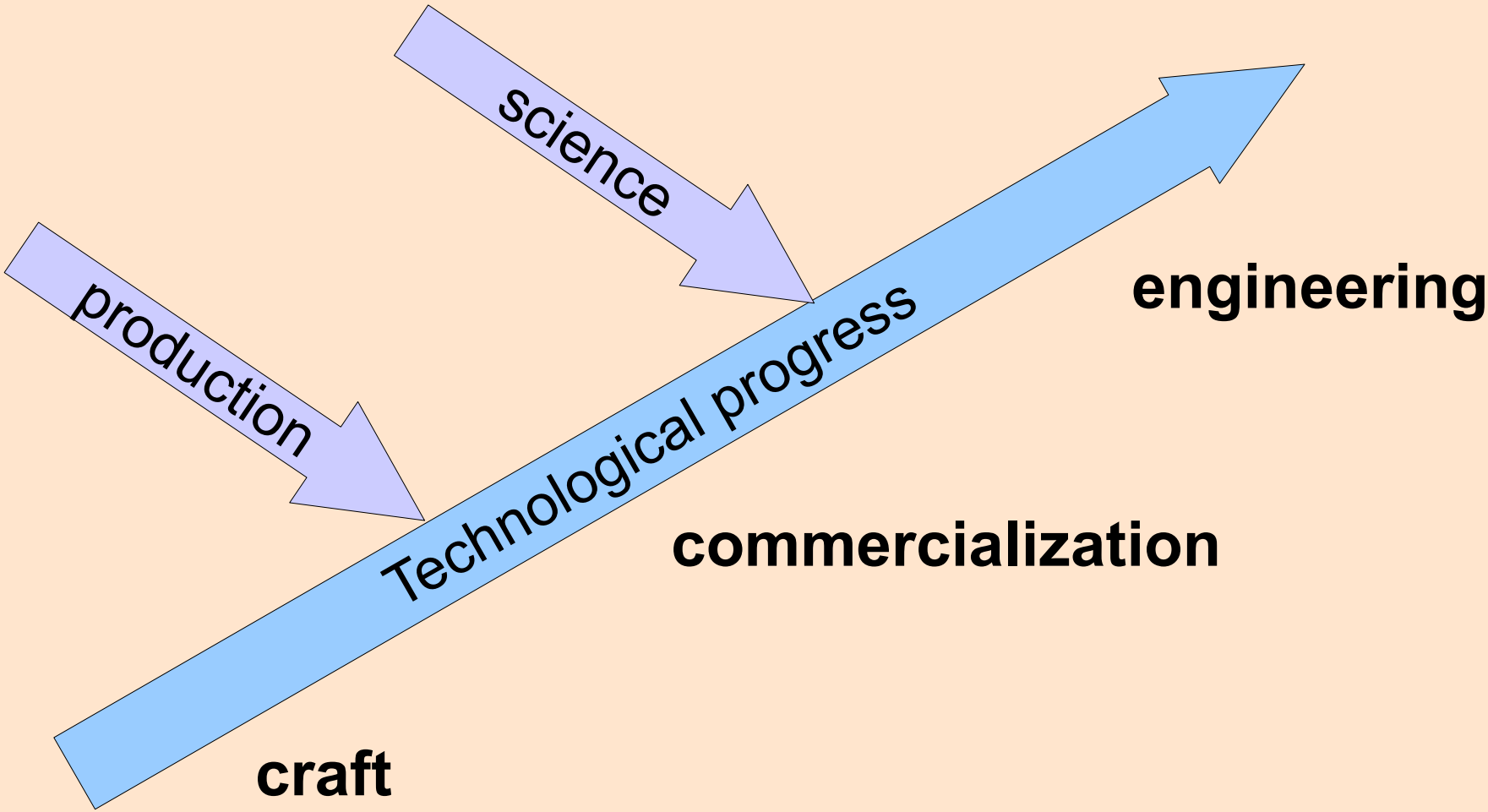
So What Is Engineering?

- Creating cost-effective solutions
- To practical problems
- By applying scientific knowledge
- To build things
- In the service of mankind

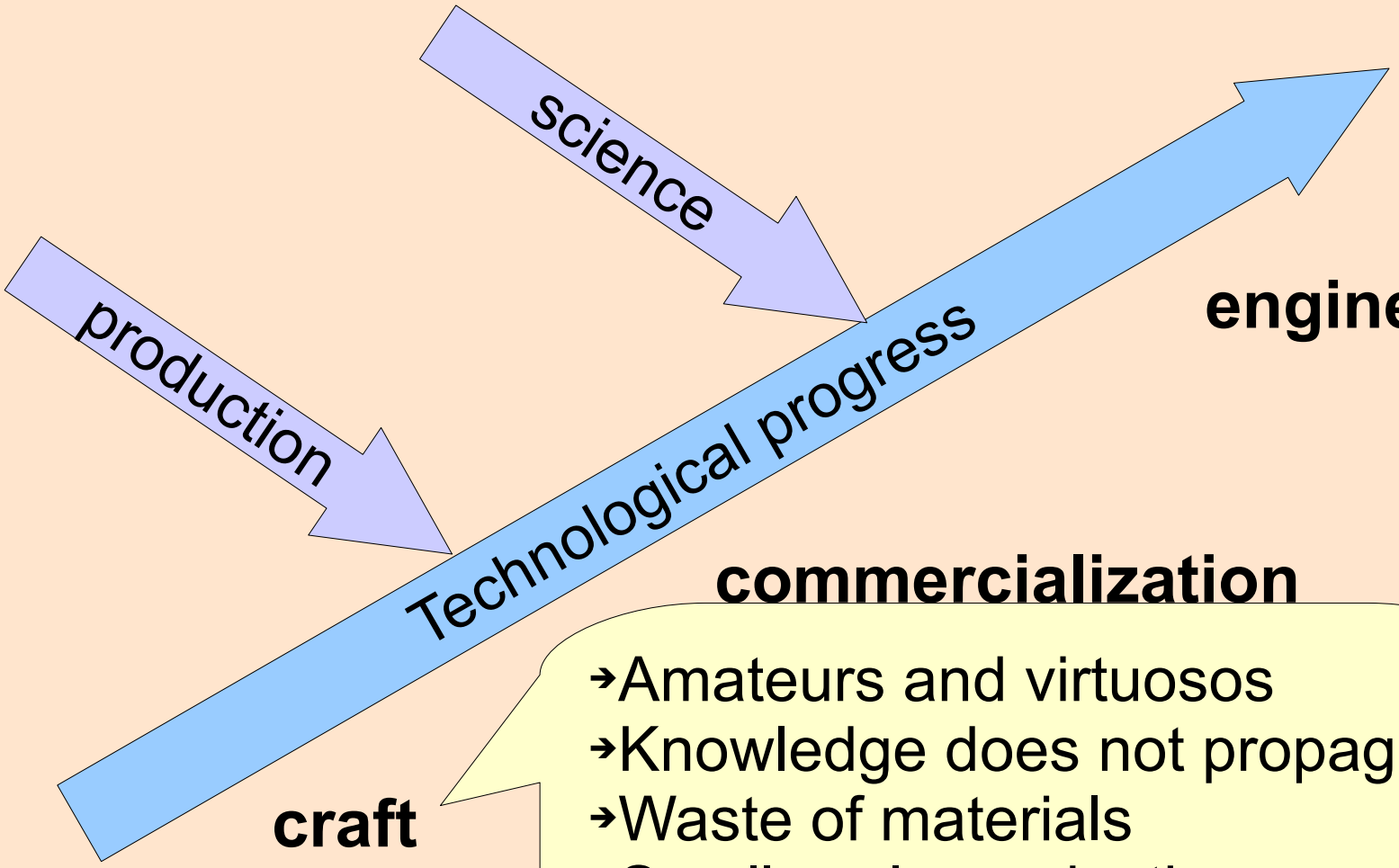
So What Is Engineering?

“Engineering relies on codifying scientific knowledge about a technological problem domain in a form that is directly useful to the practitioner, thereby providing answers for questions that commonly occur in practice. Engineers of ordinary talent can then apply this knowledge to solve problems far faster than they otherwise could. In this way, engineering shares prior solutions rather than relying always on virtuoso problem solving.”

Development of Engineering

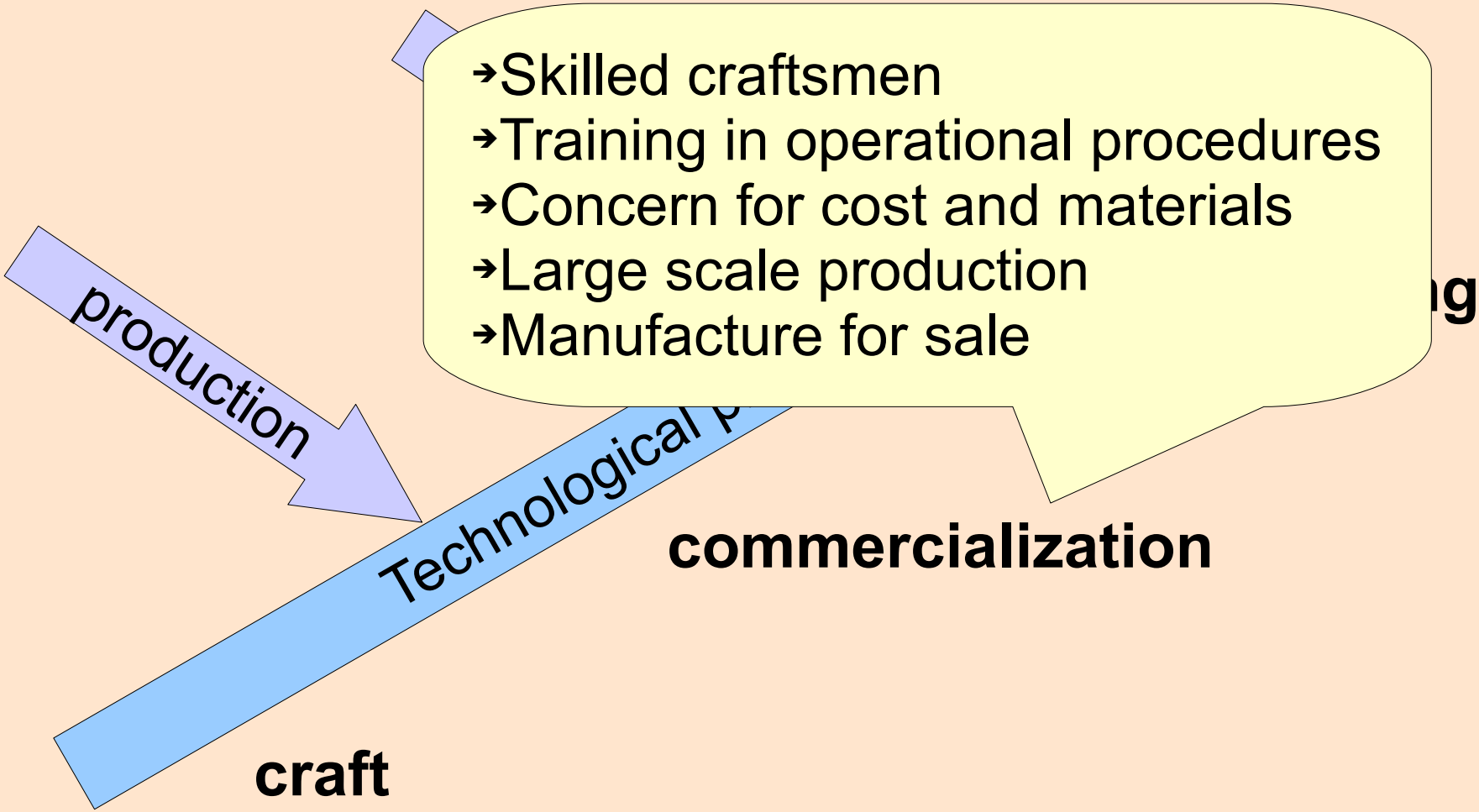


Development of Engineering

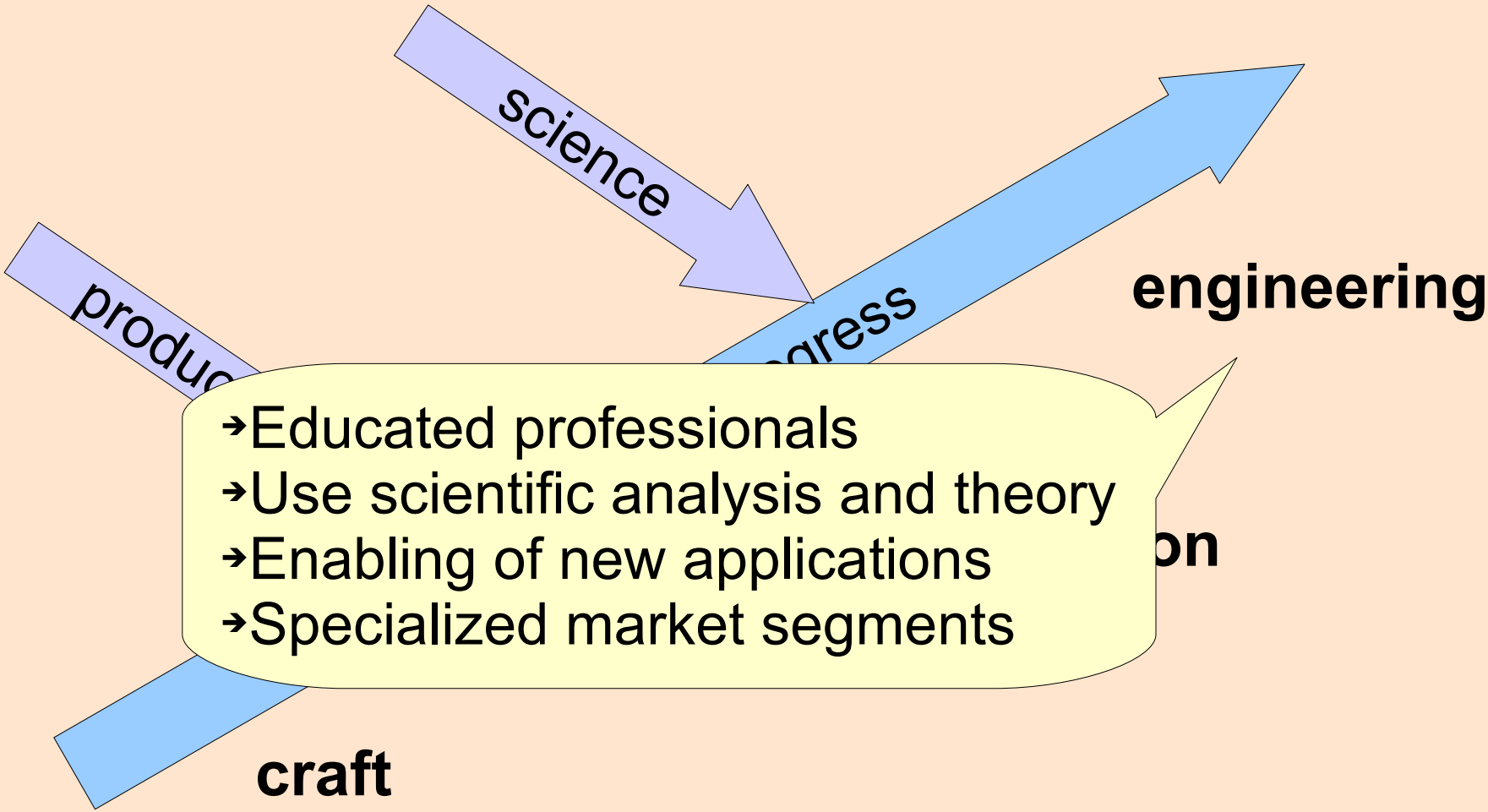


- Amateurs and virtuosos
- Knowledge does not propagate
- Waste of materials
- Small scale production
- Little commercialization

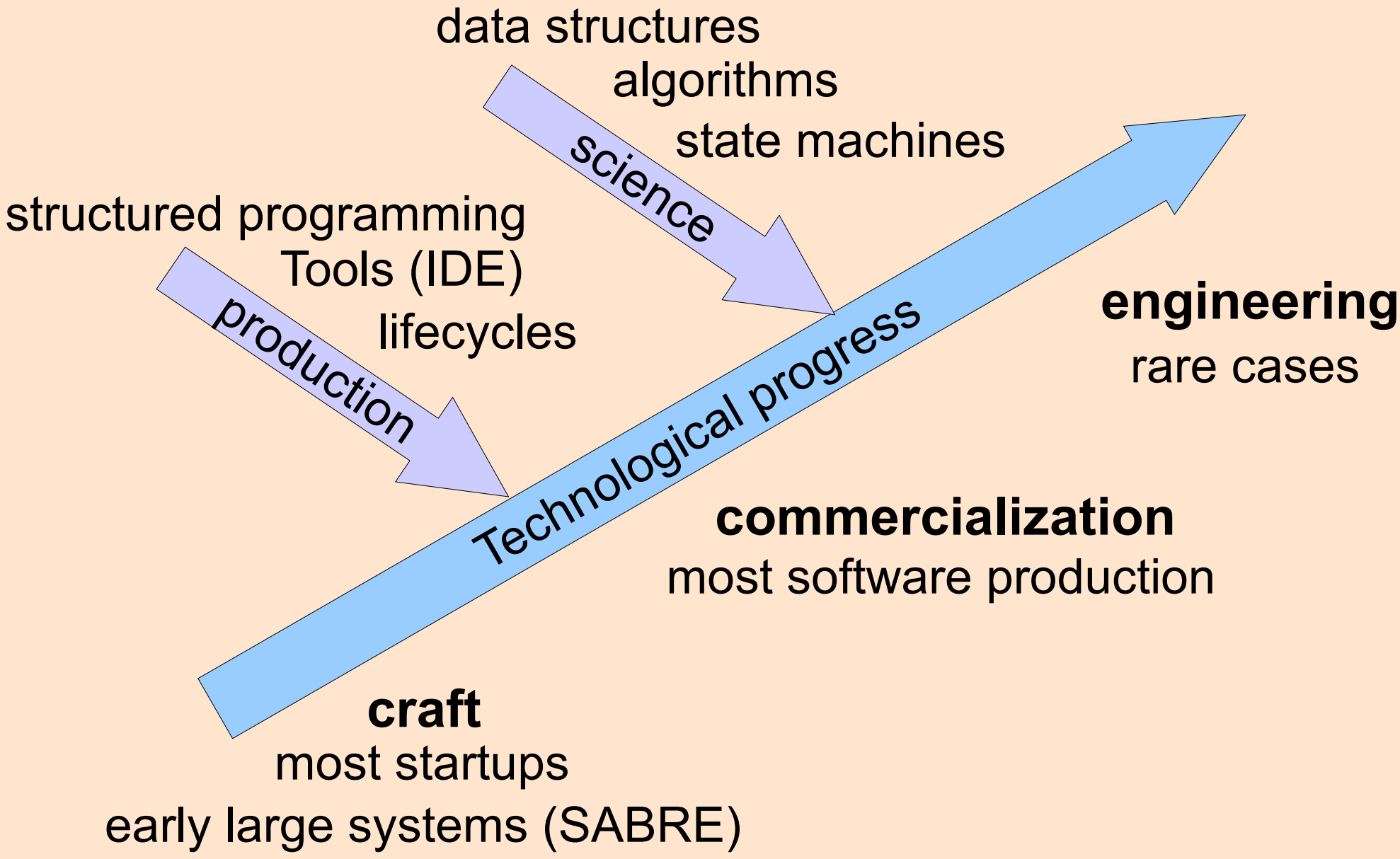
Development of Engineering



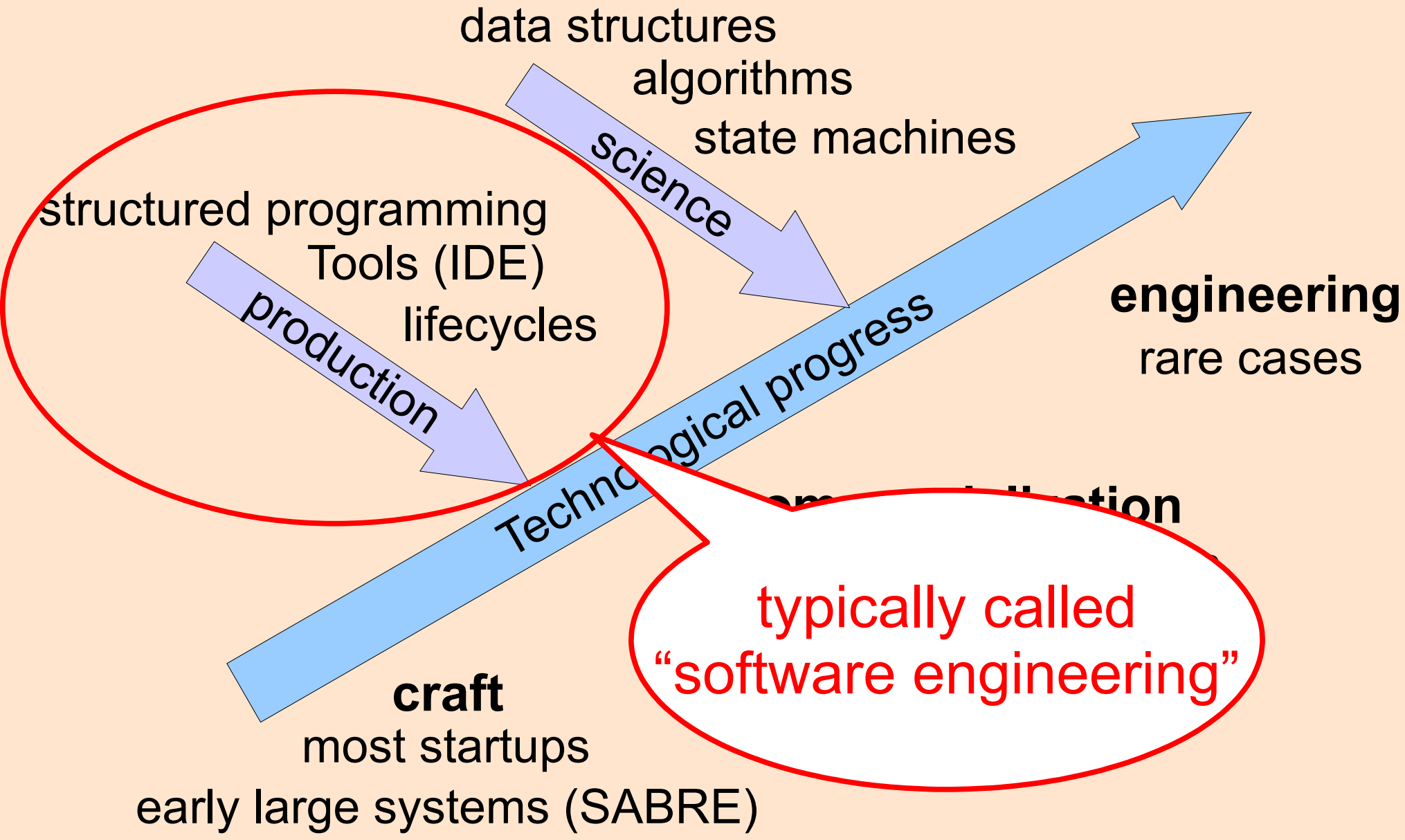
Development of Engineering



The Situation with Software



The Situation with Software



Path to True Engineering

- Define body of knowledge needed by experts
  - ➔ 50,000 chunks of information
  - ➔ 10 years of learning

Path to True Engineering

- Define body of knowledge needed by experts
- Make this knowledge accessible
  - ➔ Finding it should be easier than deriving it anew
  - ➔ Documentation of libraries etc.

Path to True Engineering

- Define body of knowledge needed by experts
- Make this knowledge accessible
- Repetition and reuse
  - ➔ Design patterns
  - ➔ Wikis and integrated environments

Path to True Engineering

- Define body of knowledge needed by experts
- Make this knowledge accessible
- Repetition and reuse
- Professional specialization
  - ➔ Nobody can master everything
  - ➔ Specialization in HCI, real-time, numerical computing, ...

Path to True Engineering

- Define body of knowledge needed by experts
- Make this knowledge accessible
- Repetition and reuse
- Professional specialization
- Improve coupling between science and commercial practice

Philippe Kruchten, “Putting the 'engineering' into 'software engineering'”. *Australian Softw. Eng. Conf.*, pp. 2-8, 2004

Developer of several large systems, e.g.
Canadian air traffic control system
Professor of SE, Univ. British Columbia
Developer of the Rational Unified Process

Software Engineering definition

According to IEEE Standard 610.12: “the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software”

- Science: unconstrained study of laws, trends, and models, with emphasis on rigor and formalism
- Engineering: perform trade-offs and compromises to make products with given level of quality under constraints of time, money, personnel, and legacy

Differentiating Characteristics

Software is different from other engineering disciplines:

- No fundamental theory
  - Computer science doesn't really help understand software
  - Compiled code is unstructured and brittle: a bug in one place causes effects elswhere
  - Software engineering limited to using best practices

Differentiating Characteristics

Software is different from other engineering disciplines:

- No fundamental theory
- Ease of change
  - Much more so than bridges etc.
  - But hard to do rigorously and take all ramifications into account

Differentiating Characteristics

Software is different from other engineering disciplines:

- No fundamental theory
- Ease of change
- Rapidly evolving technology
  - ➔ Can't consolidate body of knowledge
  - ➔ Can't benefit from many years of experience
  - ➔ Need to continuously retrain engineers

Differentiating Characteristics

Software is different from other engineering disciplines:

- No fundamental theory
- Ease of change
- Rapidly evolving technology
- Negligible manufacturing cost
  - ➔ Easy to re-deliver a fix, so no pressure to get it right the first time

Differentiating Characteristics

Software is different from other engineering disciplines:

- No fundamental theory
- Ease of change
- Rapidly evolving technology
- Negligible manufacturing cost
- No borders
  - → Easy to outsource: don't need to ship goods

Consequences I

- Waterfall model doesn't work
  - ➔ It does in other fields where things don't change
- Need to use iteration and incrementation
  - ➔ Accommodate change
  - ➔ Validate by execution and use, because theory doesn't exist

Consequences II

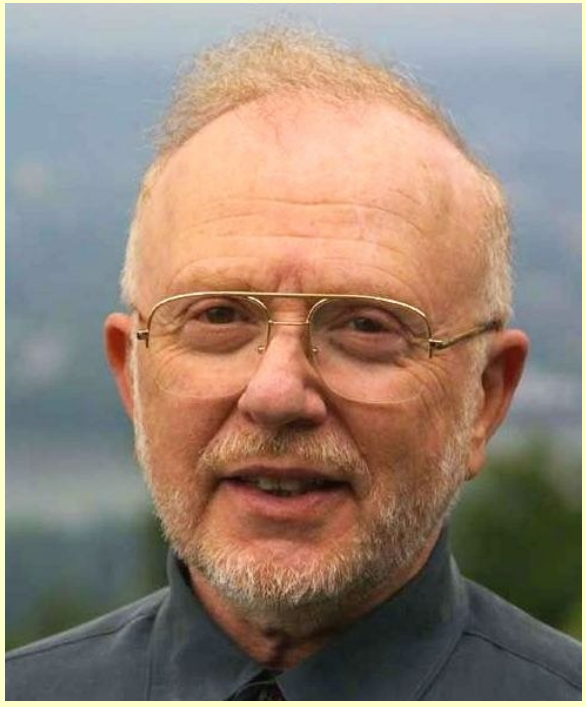
- Composability doesn't work
  - Even if components are good, we don't know whether their composition will be
  - Again due to lack of theory
  - And to the fact that technology changes rapidly
- Possibly alleviated by architecture and model-driven design

A True Profession

- Define and teach the body of knowledge
- Professional certification programs
- Liability and responsibility for products
- Shift from an inner focus (playing with technology) to an outer focus (satisfying user needs)

David L. Parnas, “Risks of Undisciplined Development”. *Comm. ACM* **53**(10), pp. 25-27, Oct 2010

Famous for idea of information hiding
Famous for criteria on dividing programs into modules
Famous for opposition to Star Wars
Professor at various universities
Fellow of ACM, IEEE



Discipline

Engineering specifies rules of action:

- Checks must be made
- Properties must be measured
- Documentation must be provided
- Test must be carried out
- Review procedures must be followed
- Inspection and maintenance routines must be followed

Engineering Education

- Prepare students for a professional career
- Meet legal requirements of their profession
  - Carry out required procedures
  - Risk being guilty of negligence if not
  - Also risk being disallowed to practice
- Fundamental principles of the field
- Not only rules but also the reasons behind them

Engineering Goals

- Produce products fit for the intended use
- Abide by applicable standards
- Be robust enough to survive foreseeable circumstances (*including wrong inputs*)
- Use conservative design with appropriate margins of error

Experience with Software

- Riddled with small petty problems
- Little if any documentation
- Meaningless error messages

The big problem:

We're so used to it we think this is how it should be

Innovation vs. Discipline

- Too much emphasis on creativity and innovation
- Too little emphasis on discipline and using check-lists

Catch-22

- As long as there is no better software, we'll buy sloppy software
- As long as we buy sloppy software, developers have no incentive to become disciplined
- If developers do not become more disciplined, they will continue to produce sloppy software

Melody M. Moore, “A License to practice software engineering”. *IEEE Software* **20**(3), pp. 112-113, May/June 2003

Interview with Leonard Tripp, Boeing Technical Fellow and past president of the IEEE Computer Society

Definitions

- Certification: passing tests to ensure you have studied a subject
- Licensing: a government service approving that you are allowed to do a certain job, typically with public safety and legal implications

Certification

- Being certified serves as testimony for competence
- Some vendors/employers may look at this favorably
- IEEE offers exam for “Certified Software Development Professional”
  - A relatively recent program
  - Only a small number certified so far

Licensing

- Does not exist in most of the world
- Does not exist in the US except the state of Texas
- In the future, expect 10-20% of software engineers to be licensed and work in health and safety related areas

Software Engineers in Israel

- As in many countries “software engineering” is not a recognized engineering discipline
  - Electrical engineering is recognized since 1960
  - Information systems engineering is recognized since 1992
  - Computer engineering is recognized since 1992
- So claiming to be a software engineer is in principle against the law