

Component-Based Software Engineering

M.R.V. Chaudron

M.R.V.Chaudron@tue.nl

www.win.tue.nl/~mchaudro/cbse2005

Technische Universiteit Eindhoven


Course Logistics

- ▶ 9 x lectures of 2 x 45 min.
 - ▶ home-work: read literature

- ▶ assignment
 - ▶ work in groups
 - ▶ design & implementation & (intermediate) report

- ▶ individual written exam

Style of the lectures

- No single good book on CBSE
- I'll provide references to relevant accessible texts
- To appreciate the lectures, read the texts
- Not only engineering/science, also
 -  I'm always interested in design rules
 - **Industrial experience / Empirical results**

References: Main Sources

Main text:

- Component Software: Beyond Object Oriented Programming
Clemens Szyperski, Addison–Wesley, (*2nd ed*, 2002)
- papers from course web page:
www.win.tue.nl/~mchaudro/cbse2004
 - Volume II: Technical Concepts of CBSE, F. Bachman et. al.,
CMU/SEI TR 2000–008, May 2000
 - Components are from Mars, Chaudron & De Jong, WPDRTS
 - Douglas Mc Illroy, 1968/9, NATO Conference on SE

References: Background Literature

- Component-Based Development: Principles and Planning for Business Systems, Katharine Whitehead, Addison-Wesley 2002
Orientation towards business systems.
- Building Reliable Component-based Software Systems, Ivica Crnkovic & Magnus Larson (eds), Artech House Publishers, 2002
Orientation towards technical, embedded, Real-Time systems
- UML Components: A Simple Process for Specifying Component-based Software, Cheesman & Daniels, Addison-Wesley, 2001
Heavily leans on UML, describes some design guidelines for business information systems
- [Building Systems from Commercial Components](#), Kurt C. Wallnau, Scott A. Hissam, and Robert C. Seacord, Addison-Wesley (SEI series), 2001

Lecture Scheme

1. Introduction, Motivation, Concepts, Terms & Defs
2. Design of Component Models & Components
3. Reuse & Economics of Reuse
4. Specification of Components
5. Industrial Example of CBD – Koala
6. Component Testing & Certification
7. Generative Programming
8. Sw Composition & Predictable Assembly
9. t.b.d. / aAdvanced topics

Selection of topics from the CBSE area
Some more stable than others.

Who is Michel Chaudron?

- Teach:
 - Software Architecting (1st year M.Sc.)
 - Component-Based Software Engineering (3rd year, B.Sc.)
- Research:
 - Member of the System Architecture and Networking (SAN) Group
 - Subjects: CBSE (predicable assembly), Sw. Arch., Empirical SE
 - How can we predict extra-functional properties of the systems we design ?
 - Collaboration with industry (ASML, Nokia, Oce, Philips Natlab, Medical, ...) via a number of European Research Projects:
- Coordinates: HG 5.03, m.r.v.chaudron@tue.nl; www.win.tue.nl/~mchaudro
- secretary: mw. Cecile Brouwer, HG 5.08, tel. 247 8309

Today's Lecture

- ▶ Introduction CBSE & Reuse
 - ▶ Motivation
 - ▶ Concepts, Definitions, Terminology

Observations on the practice of SE

About 80% of software engineering deals with changing existing software

It is not the strongest of the species that survive, nor the most intelligent, but the ones most responsive to change. -- Charles Darwin

Time to market is an important competitive advantage: incorporate successful innovations quickly

- Systems should be built to facilitate change
 - easy removal and addition of functionality

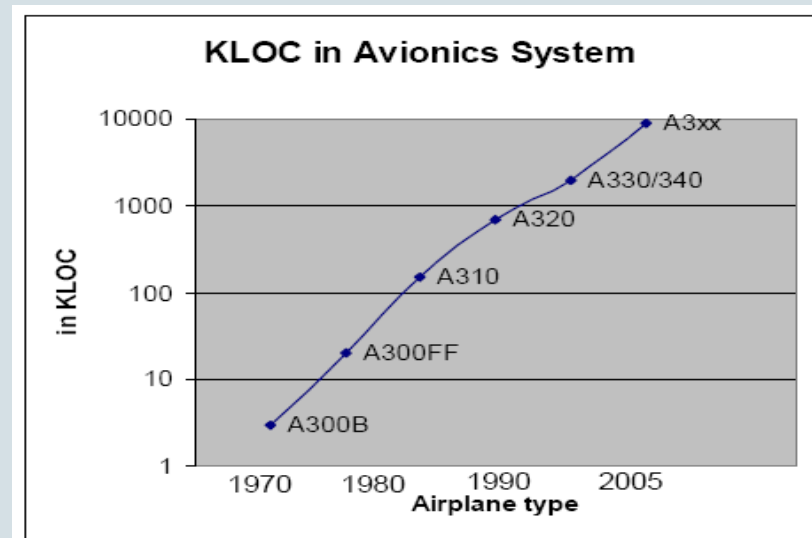
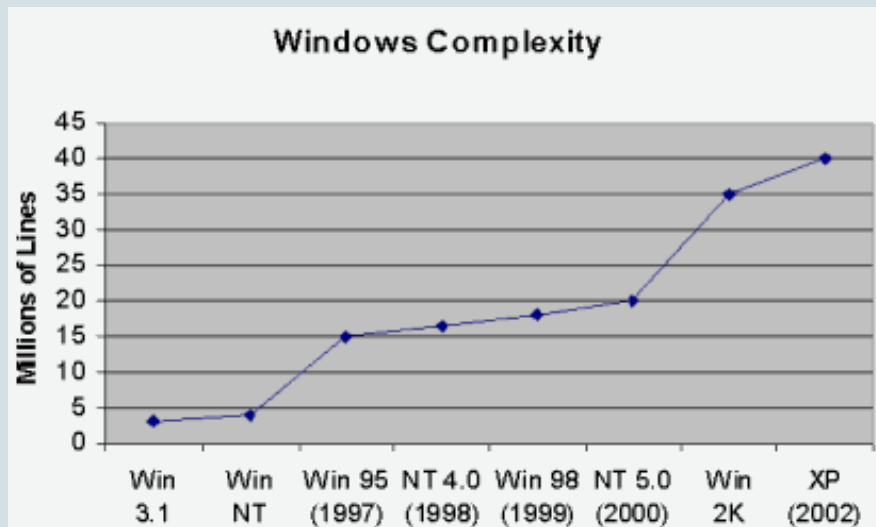
"Most software today is very much like an Egyptian pyramid with millions of bricks piled on top of each other, with no structural integrity, but just done by brute force and thousands of slaves"

Alan Kay, creator of SmallTalk

Problems of Software Engineering

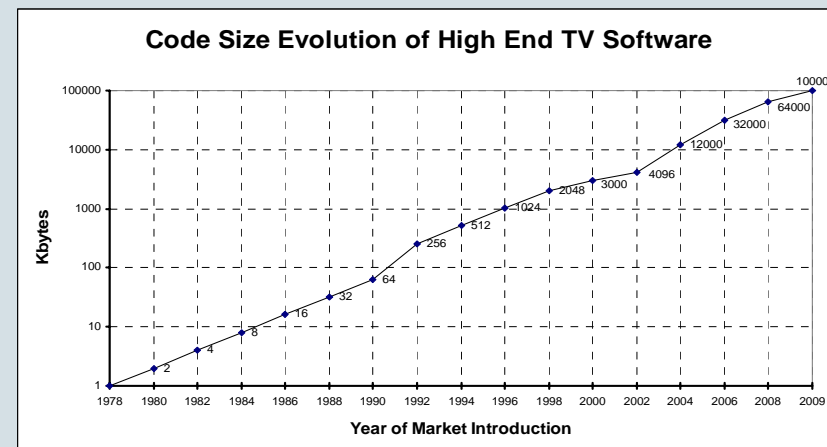
- The size & complexity of software increases rapidly
- Single products become part of product families
- Software is upgraded after deployment
- The time-to-market must decrease significantly
- The cost of products must be reduced

Increasing amount of software in systems

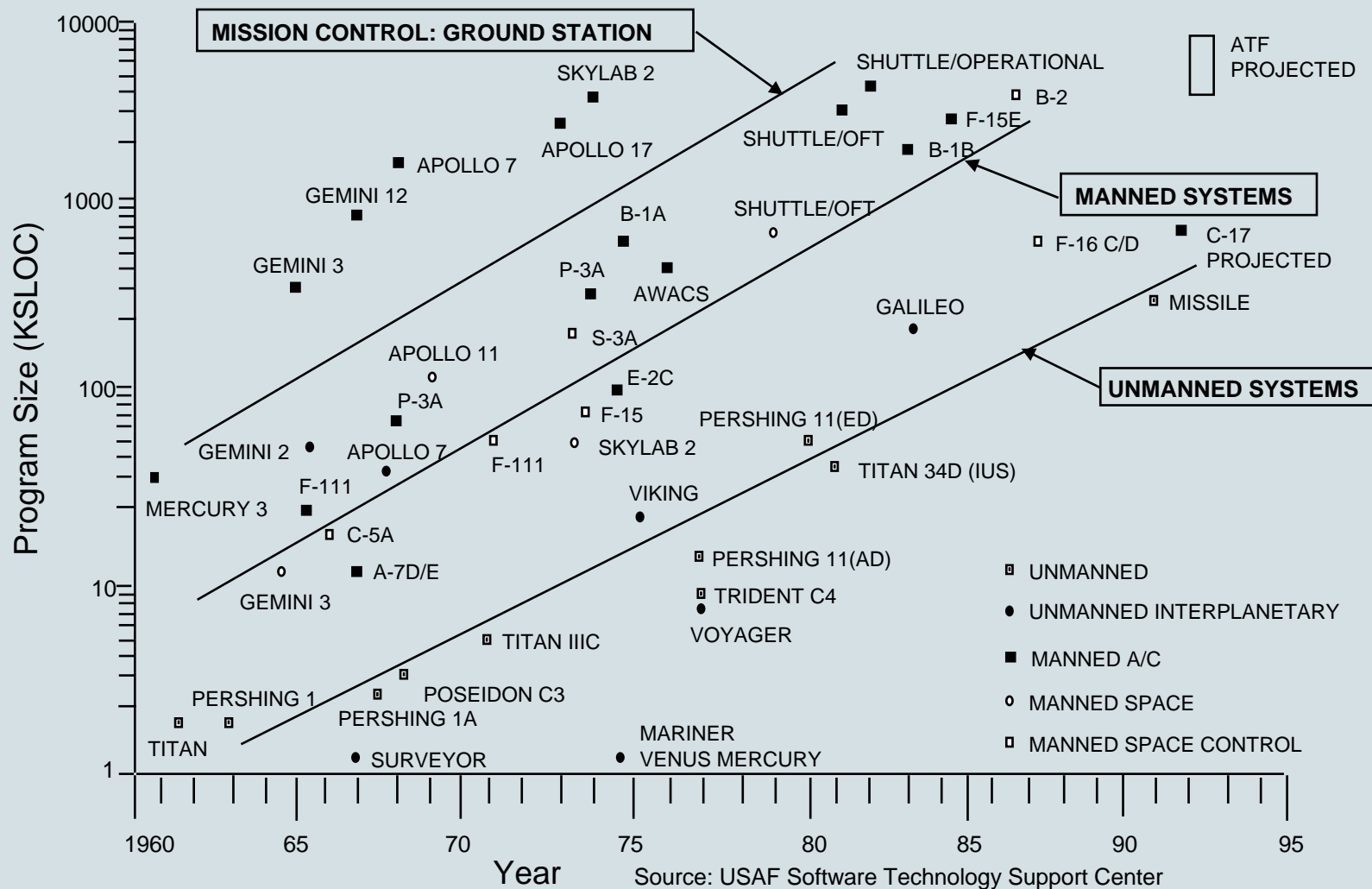


Nb: logarithmic scale

The Amount of software increases

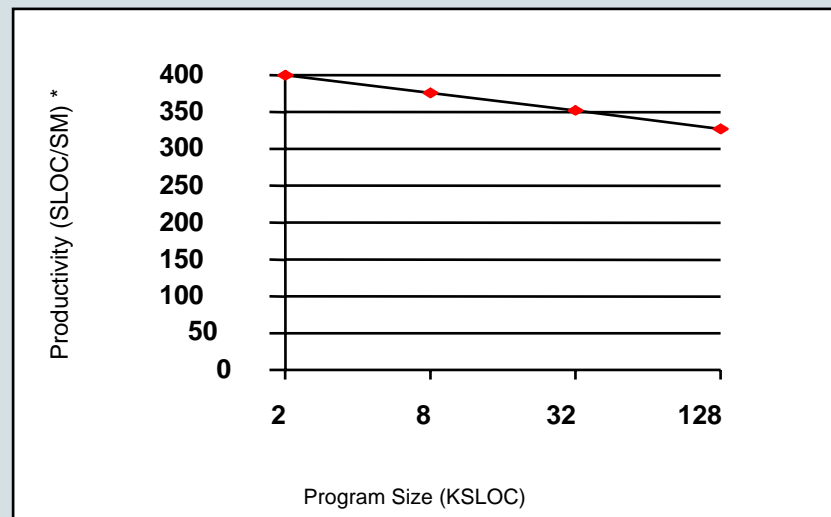


SOFTWARE GROWTH

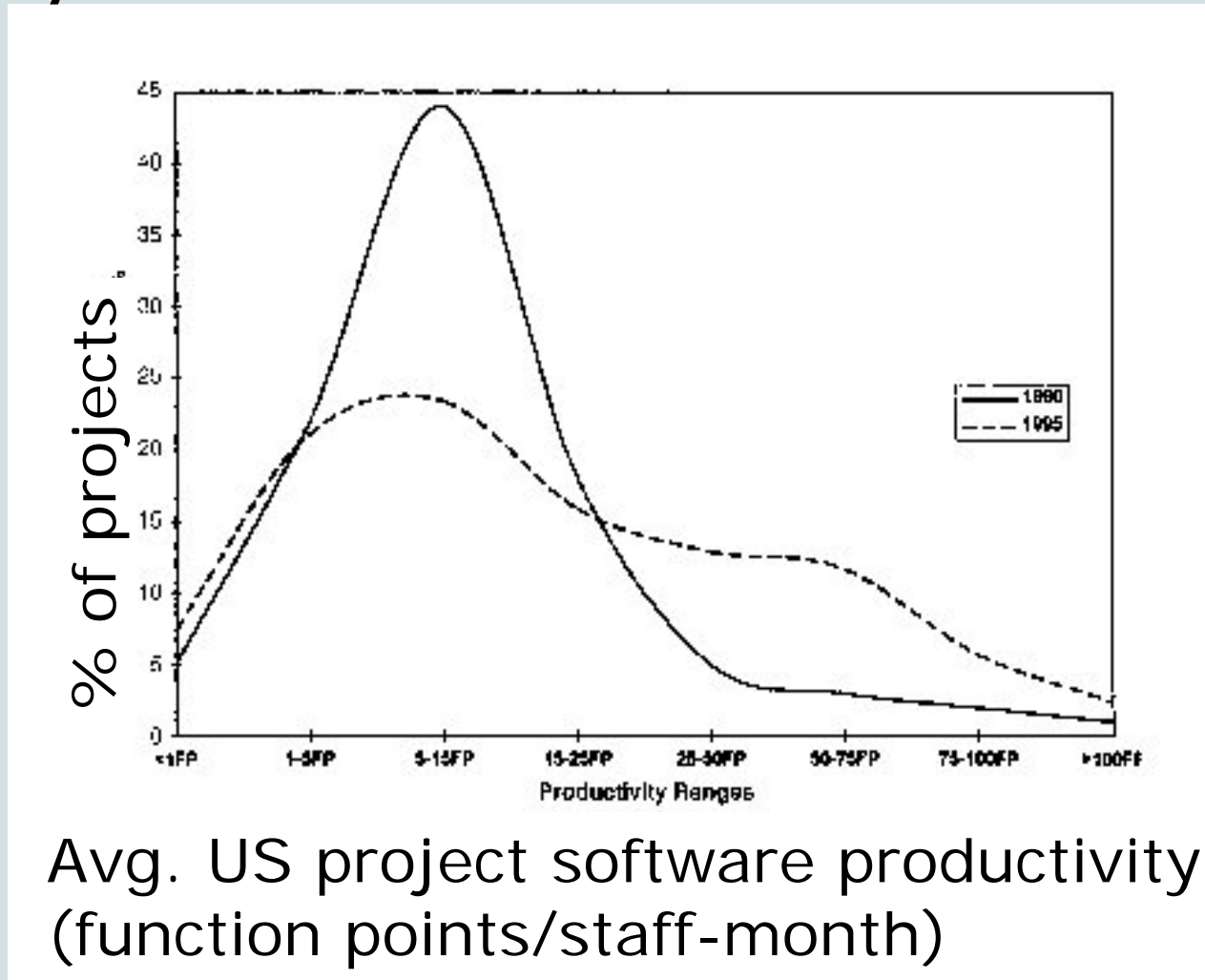


Diseconomy of Scale

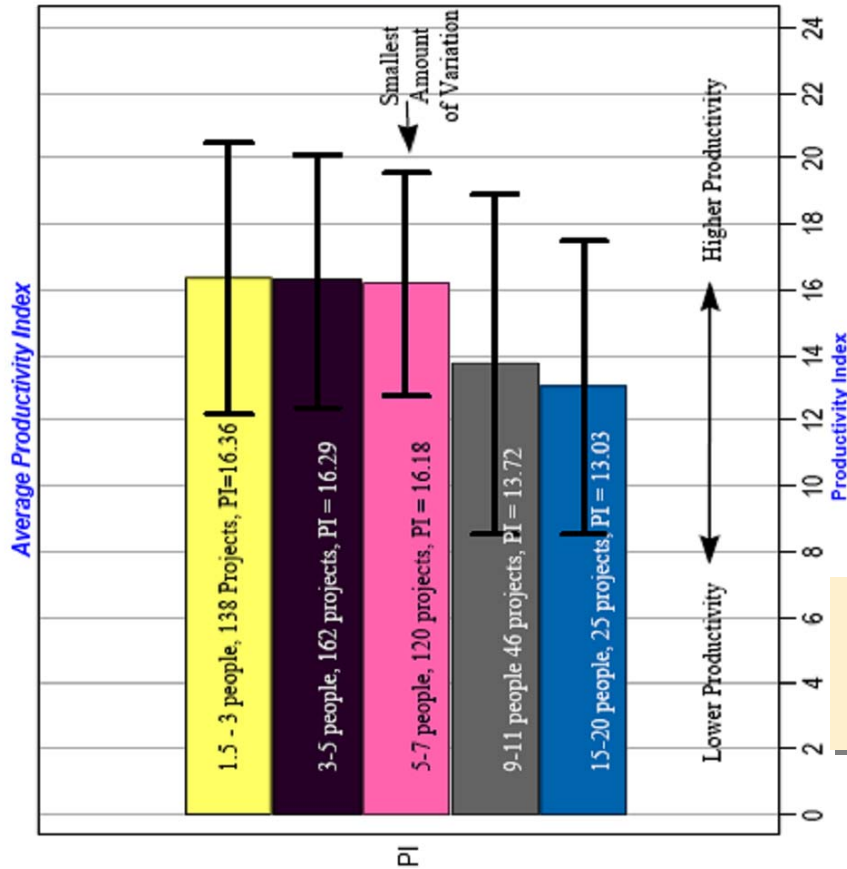
- Barry Boehm (Source: Software Engineering Economics)
 - “The more members that are added to a team, the more . . . time is consumed in communication with other team members . . .”
 - “This decrease in [programmer] productivity on larger projects is called a **diseconomy of scale** in economic terms.”
- Methods with different impacts on productivity/cost will have a relatively greater impact as program size increases



Survey of Distribution of Productivity



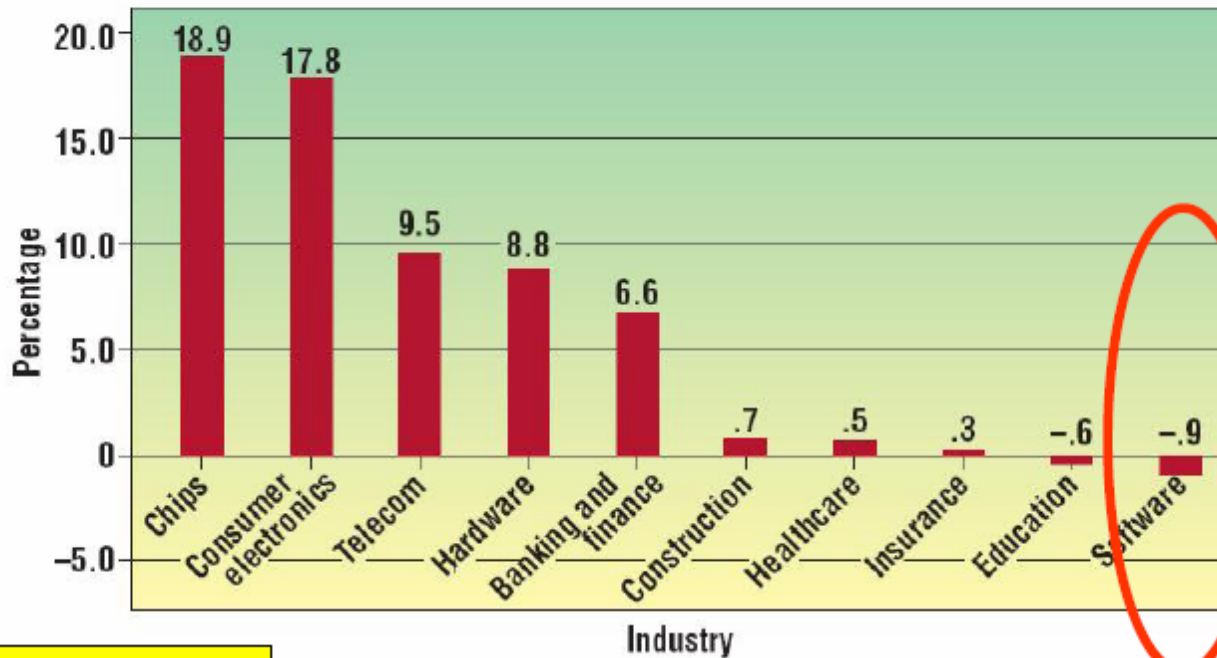
Productivity & Team Size



Throwing more people at the problem, reduces productivity

From: Familiar Metric Management – Small is Beautiful–Once Again
 By Lawrence H. Putnam and Ware Myers

Annual Productivity Growth (1998–2003)



Fact corner:
New innovations are
needed

Source: <http://www.economy.com>

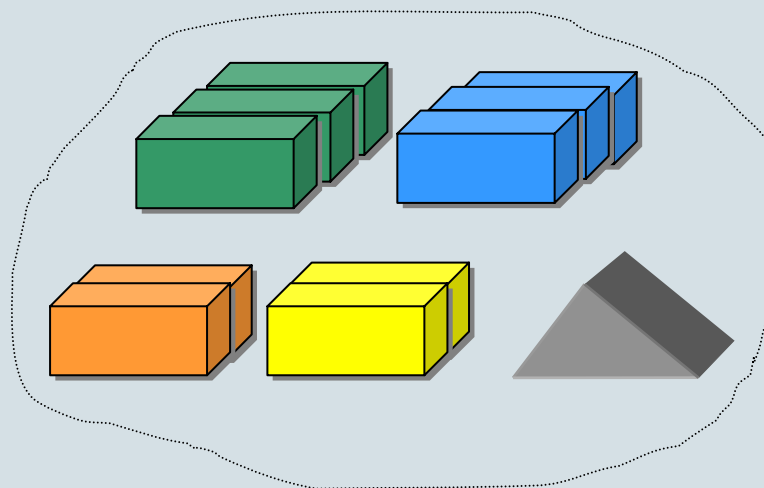
The CBD-‘Solution’

Systems should be assembled from existing components

Idea dates (at least) to the 1968 NATO Conference

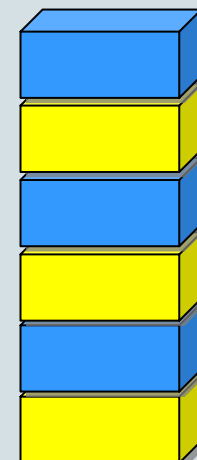
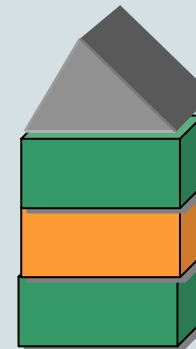
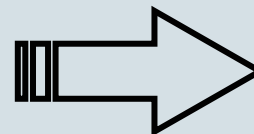
Douglas McIlroy: Mass Produced Software Components

component repository & market



component-based systems

compose



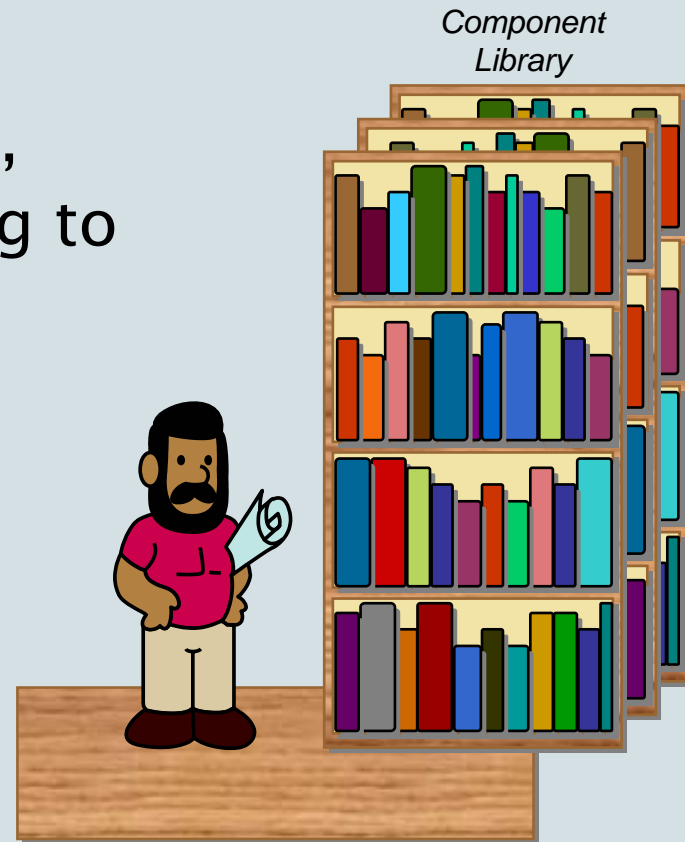
Why Components?

Following other engineering disciplines (civil and electrical), software engineering is looking to develop

a catalogue of software building blocks

connection standards

Confusing or helpful?



What is CBSE?

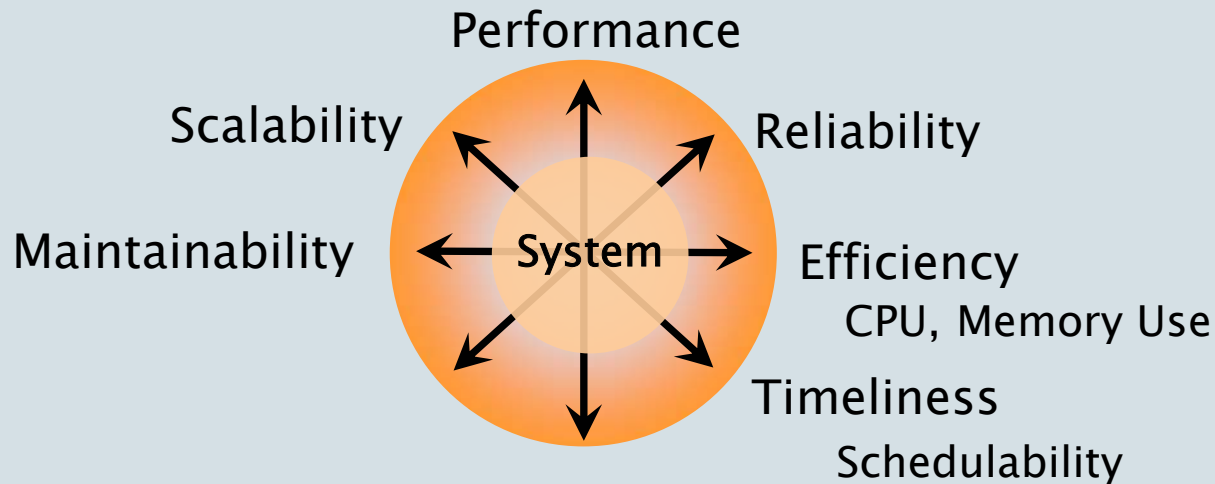
based on definition of SEI in CMU/SEI-2000-TR-008

Component-based Software Engineering is concerned with the *rapid assembly* and *maintenance* of component-based systems, where

- components and platforms have *certified properties*
- these certified properties provide the basis for *predicting properties of systems* built from components.

Predictability is a *key* property of mature engineering disciplines. It enables feedback on design and adaptation; i.e. development time is reduced because we can analyze prior to building

Extra Functional Properties



Essential system engineering problem:

- a plurality of contradictory goals
- a plurality of means (technology, process)
each of which provides a varying degree of help or hindrance in achieving a given goal

Business Drivers for CBD

Improve Productivity:

Build more software using fewer resources through enabling the assembly of systems from components that may be independently developed by different parties.

Independent in time and space

- independent from ultimate application
- independent from (future) peer-components

Motivations for CBSE

- Productivity
- Quality
- Time-to-market
- Maintenance

Strategic business goals that increase

- Turnover
- Market share

reduce

- Cost of development
- Cost of ownership

€ € €
profit
€ € €

CBSE & Software Productivity

Increase competitiveness (sw/€):

- Reduce cost of development
- Increase software/€

Limited human talent (sw/people):

- Increase software/person
 - ⇒ reuse existing solutions, rather than invent them

CBSE & System Quality

Improve Quality:

Idea: Assuming that a collection of high-quality components is available, assembling these should yield systems of high-quality.

1. The cost of establishing the high quality of components is amortized over multiple use.
2. Multiple use of a component increases the likelihood of finding and removing errors.

CBSE & Maintenance

The use of CBD requires good modular design.

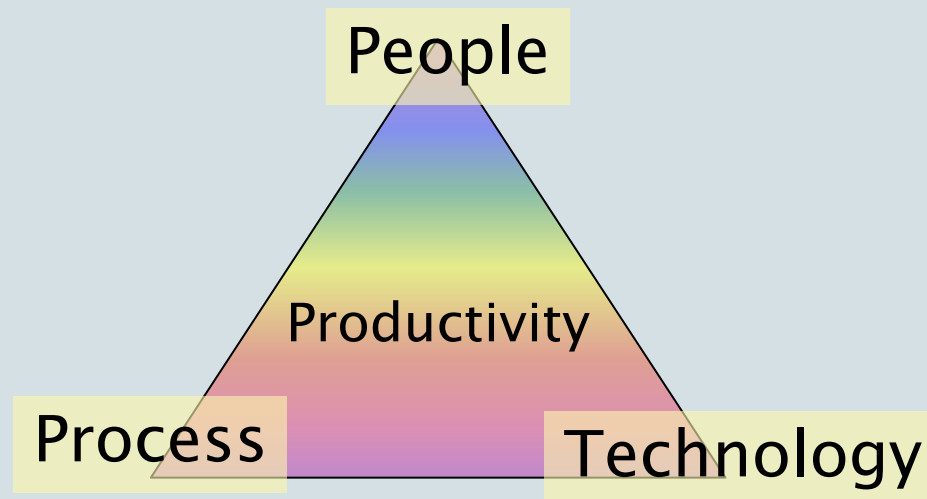
This modularity provide quality properties like

- comprehensibility/understandability
- maintainability
- flexibility
- ...

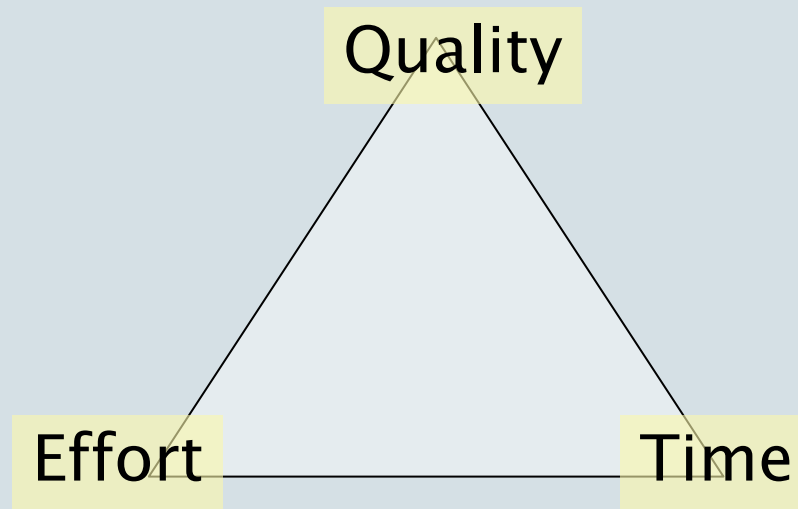
CBSE & Time-to-market

If the reuse of a component requires less time than the development of a component, systems can be built faster.

Productivity & Quality



Dependent dimensions



Technical Drivers for CBSE



CBSE may help improve system qualities

Reuse-based Software Engineering

- Reuse-based SE has many business drivers in common with CBSE:
 - increase productivity & quality
 - reduce time-to-market,
 - reduce development cost

However, reuse imposes less technical- and design-constraints on the unit of reuse (*asset*).

CBSE enables Reuse, Reuse is not sufficient for CBSE.

Reusable Assets

Virtually any product of the SE process can be reused:

- Requirements
- Architectures
- Designs
 - design patterns, interfaces
- Source Code
 - ranging from to libraries, patterns, to modules, to macros, coding conventions, ...
- Test Scripts

Questions?

What is a software component?

How can you recognize a software component?

Suggestions from the audience?

Reflect on differences between civil and electrical engineering on the one hand and software engineering on the other hand

Cross-cutting concerns

What is a Component?

Probably more definitions than for ‘software architecture’

A software component is a unit of composition with contractually specified *interfaces* and *explicit context dependencies* only

A software component is *independently deployable* and subject to *composition* by third parties.

Clemens Szyperski, 1997

What is *independent deployment*?

A software component is a unit of *independent deployment*

- no dependencies on peer-components
 - some ‘meaningful’ functionality by itself
 - components tend to be ‘large grained’

- never partially deployed

What is a Component?

A reusable software component is a logically cohesive, loosely coupled module that denotes a single abstraction.

Grady Booch, *Software Components with Ada, 1987*

Tries to provides some design guidance.

What is cohesive? loosely coupled? single abstraction?

Object Technology and CBSE

“OT is Neither Necessary Nor Sufficient for CBSE”

OT was a useful and convenient starting point for CBSE

OT did not express full range of abstractions needed by CBSE

(insufficiency)

It is possible to realize CBSE without employing OT(*non-necessity*)

CBSE might induce substantial changes in approach to system design, project management, and organizational style

What is a Component?

“A binary unit of independent production, acquisition, and deployment that interacts to form a functioning system.”

– C. Szyperski, *Component Software*

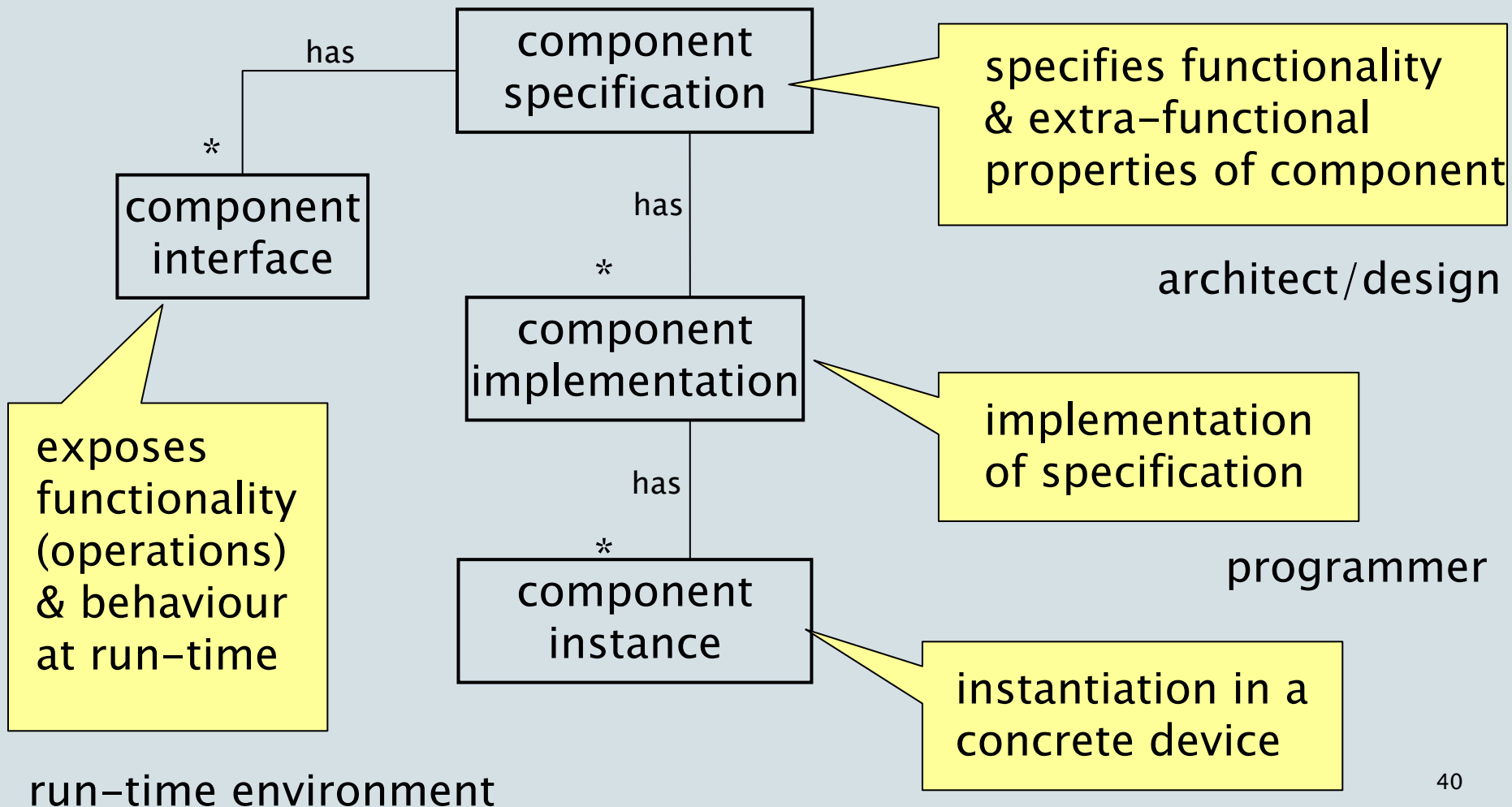
“A component is an independently deliverable package of operations.”

– *Texas Instruments Literature*

“A replaceable unit of development work which encapsulates design decisions and which will be composed with other components as part of a larger unit.”

– Desmond D’ Souza, in *Catalysis*

Useful Distinction



It depends (on what?)

It depends to some degree on what you want to do with components:

- ▶ REPLACING one part of an implementation with another:
 - then adhering to a well defined specification is sufficient
 - at run-time, then mechanisms are needed for
 - registering, locating / binding components
- ▶ EXTEND components
 - extend interfaces, merge implementations

It depends (on what?)

- ▶ TRADING (use across multiple organizations)
 - need mechanism for
 - hiding intellectual property
 - certification

- ▶ RE-USE
 - need repository for searching / matching, versioning
 - economics impact granularity

- ▶ Application domain BUSINESS/EMBEDDED
 - type and granularity of component

Summary

- CBD aims to earn money through improving Productivity, Quality, Time-to-market, Maintenance
- CBSE is about the assembly of components
- CBSE enables Reuse; Reuse is not necessarily about components

Questions?

You should know

- an answer to 'What is a component?'
- the relation between reuse, CBD, and OO

Self-Study material:

- paper by Doug McIlroy: Mass Produced Software Component
- Tech. Report SEI: Technical Concepts of CBD