LUT University

LAND OF THE CURIOUS

LECTURE 2

INTRODUCTION – SOFTWARE ENGINEERING

LUT University

COURSE LECTURE SCHEDULE

Date	Topic	Book Chapter(s)
Wed 8.9.	Course introduction	
Tue 14.9.	**Introduction to Software Engineering**	**Chapter 1**
Tue 21.9.	Software Processes	Chapter 2
Mon 27.9	Agile Software Engineering	Chapter 3
Tue 5.10.	Requirements Engineering	Chapter 4
Mon 11.10.	Architectural Design	Chapter 6
Wed 20.10.	Modeling and implementation	Chapters 5 & 7
Mon 1.11.	Testing & Quality	Chapters 8 & 24
Mon 8.11.	Software Evolution & Configuration Management	Chapters 9 & 25
Mon 15.11.	Software Project Management	Chapter 22
Mon 22.11.	Software Project Planning	Chapter 23
Mon 29.11.	Global Software Engineering	
Wed 8.12.	Software Business	
Mon 13.12.	Last topics	

Teachers:
Maria
Susan
Sami Hyrynsalmi

GOALS FOR THIS LECTURE:

After this lecture, you know

1. What is software engineering?
2. Why is it important?
3. What kind software engineering jobs exist
4. Why is software engineering challenging?

HONOR CODE

- <u>You must accept the honor code to take the course</u>
- Read Moodle for the details
- In practice:
  - You must do all assignment by yourself, without help or collaboration with other students, unless explicitly allowed or required
  - You must write your own assignments, and take the quizzes by yourself, without discussing them with other students until you have submitted your own and the other students have submitted theirs
  - Some activities may require collaboration (e.g. project), if you are required to do something in small teams, it will be mentioned separately
  - You must **<u>reference</u>** any sources you use, including the course textbook

WHAT IS SOFTWARE ENGINEERING?

Software engineering

is an engineering discipline that is concerned with all aspects of software production from the
early stages of system specification through to
maintaining the system after it has gone into use.

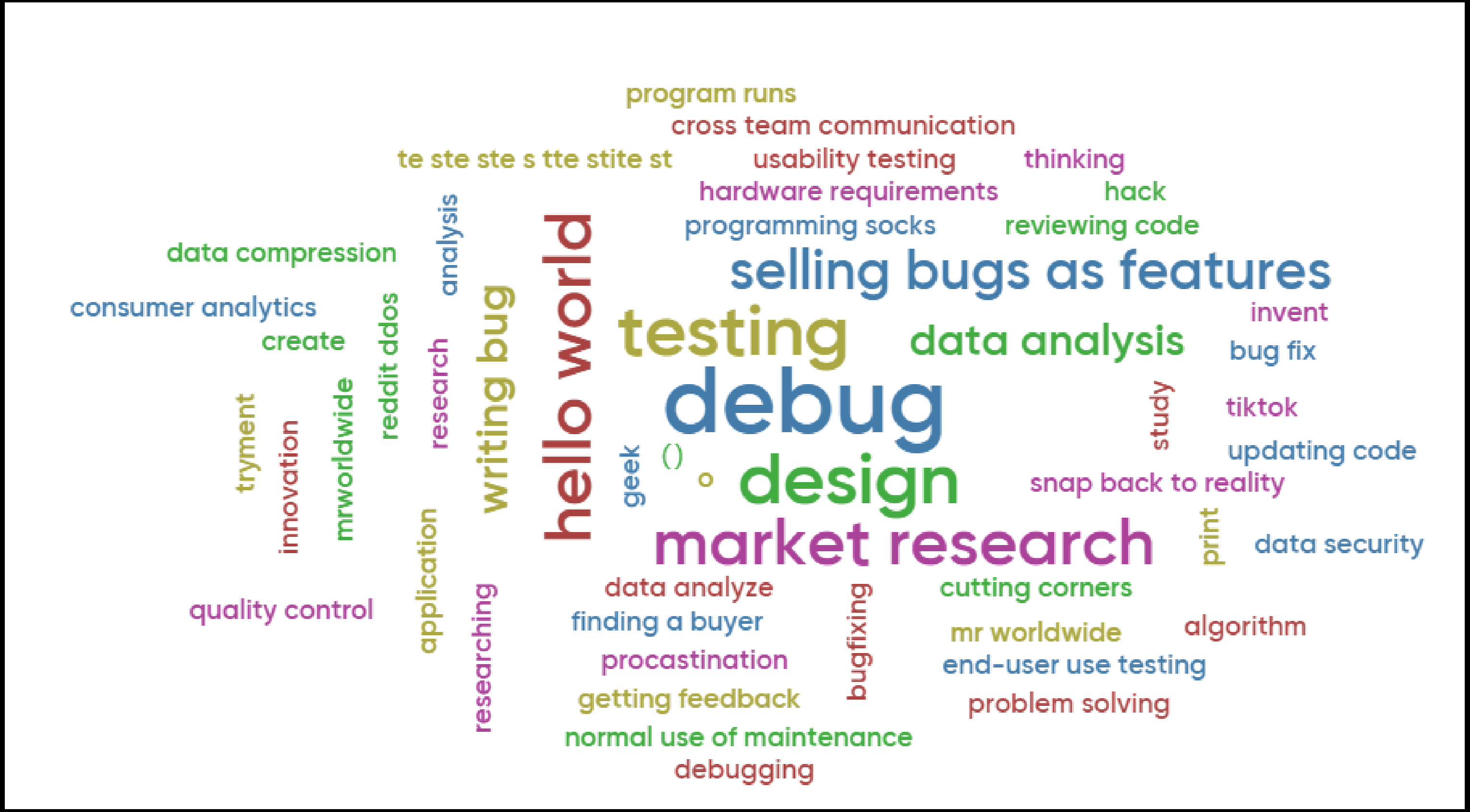
Software Engineering is

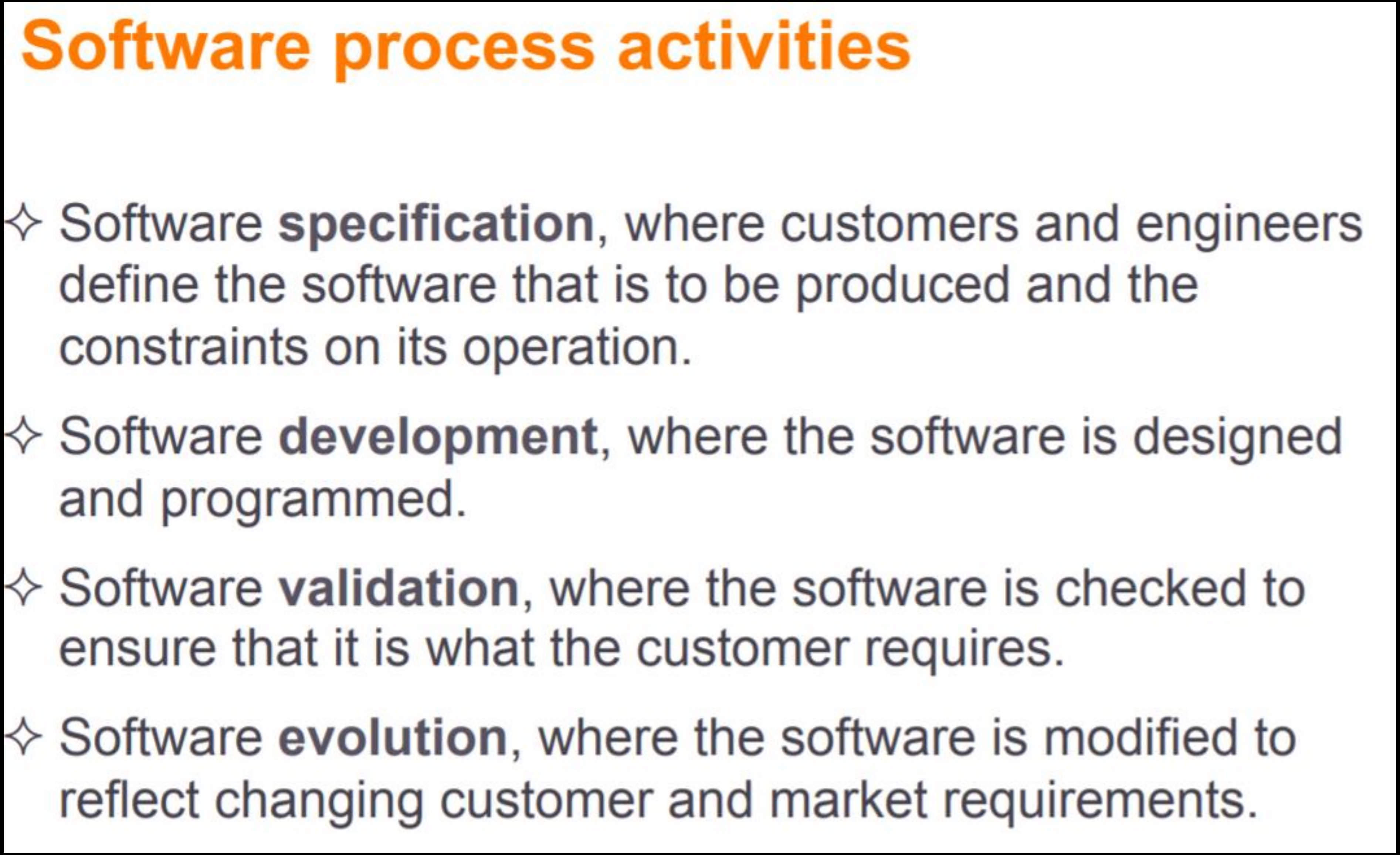
… is an engineering discipline that is
concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.

- Engineering discipline:
Using appropriate theories and methods to solve problems & bearing in mind organizational and financial constraints.

- All aspects of software production: Not just the technical process of development. Also project management and the development of tools, methods etc. to support software production.

DURING THE LAST LECTURE: YOU WROTE PHASES AND ACTIVITIES THAT ARE PART OF SW ENGINEERING





Software process activities

- ✧ Software **specification**, where customers and engineers define the software that is to be produced and the constraints on its operation.
- ✧ Software **development**, where the software is designed and programmed.
- ✧ Software **validation**, where the software is checked to ensure that it is what the customer requires.
- ✧ Software **evolution**, where the software is modified to reflect changing customer and market requirements.

Software engineering: Why?

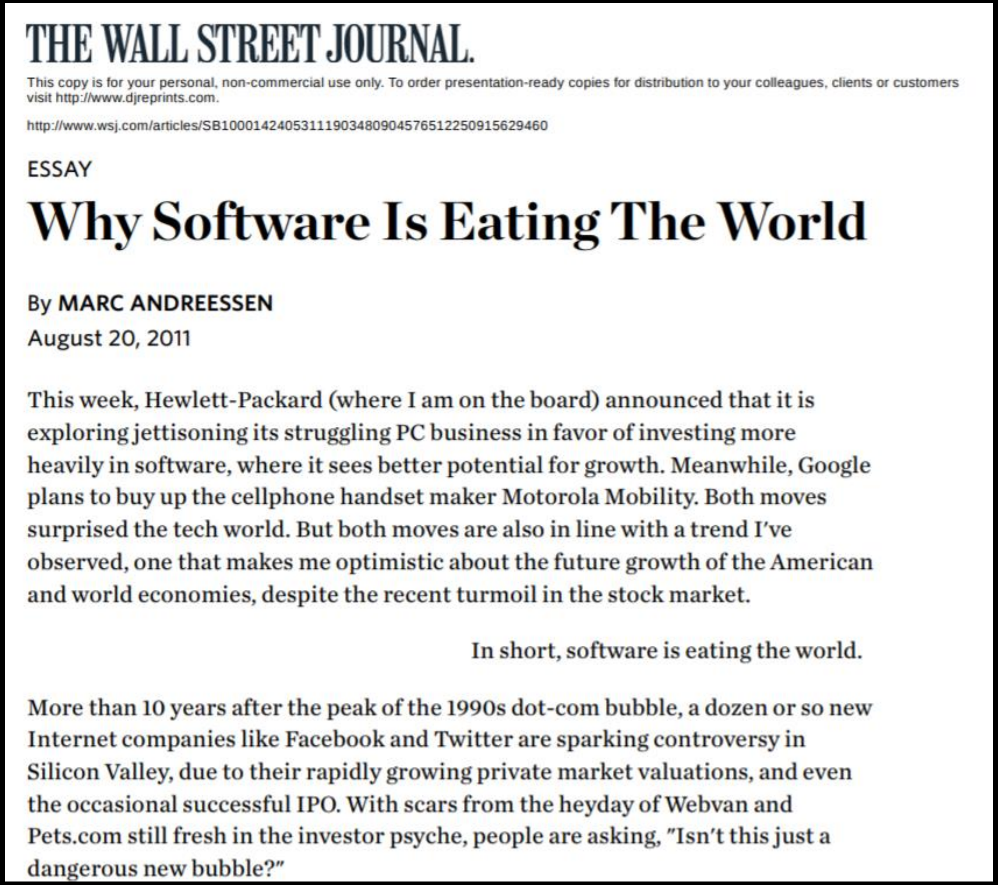
- ✧ The economies of all developed nations are dependent on software.
- ✧ More and more systems are software controlled
- ✧ Software engineering is concerned with theories, methods and tools for professional software development.
- ✧ Expenditure on software represents a significant fraction of GNP in all developed countries.

Software costs

- ✧ Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- ✧ Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- ✧ Software development cost are 60% and testing costs are 40% of the software costs
- ✧ Software engineering is concerned with cost-effective software development.

WHY SOFTWARE IS EATING THE WORLD

- The Wall Street Journal, Aug, 2011, By Marc Andreessen
- Software companies are taking over large parts of the economy -> major industries and businesses are run on software
- Software is “eating” some of the traditional business
  - E.g. Online book sales: Borders vs. Amazon
- Music companies are software companies
  - E.g. Spotify, iTunes, Pandora
- Movie production companies are software companies
  - E.g. Pixar that was bought by Disney
- Electric cars are computer controlled, retail logistics and logistic chains are optimized with software
- Banks have huge software development departments
- National social security programs have sw departments
- Job market is changing: lack of education and skills in software area



THE WALL STREET JOURNAL.

This copy is for your personal, non-commercial use only. To order presentation-ready copies for distribution to your colleagues, clients or customers visit http://www.djreprints.com.

http://www.wsj.com/articles/SB10001424053111903480904576512250915629460

ESSAY

Why Software Is Eating The World

By MARC ANDREESSEN
August 20, 2011

This week, Hewlett-Packard (where I am on the board) announced that it is exploring jettisoning its struggling PC business in favor of investing more heavily in software, where it sees better potential for growth. Meanwhile, Google plans to buy up the cellphone handset maker Motorola Mobility. Both moves surprised the tech world. But both moves are also in line with a trend I've observed, one that makes me optimistic about the future growth of the American and world economies, despite the recent turmoil in the stock market.

In short, software is eating the world.

More than 10 years after the peak of the 1990s dot-com bubble, a dozen or so new Internet companies like Facebook and Twitter are sparking controversy in Silicon Valley, due to their rapidly growing private market valuations, and even the occasional successful IPO. With scars from the heyday of Webvan and Pets.com still fresh in the investor psyche, people are asking, "Isn't this just a dangerous new bubble?"

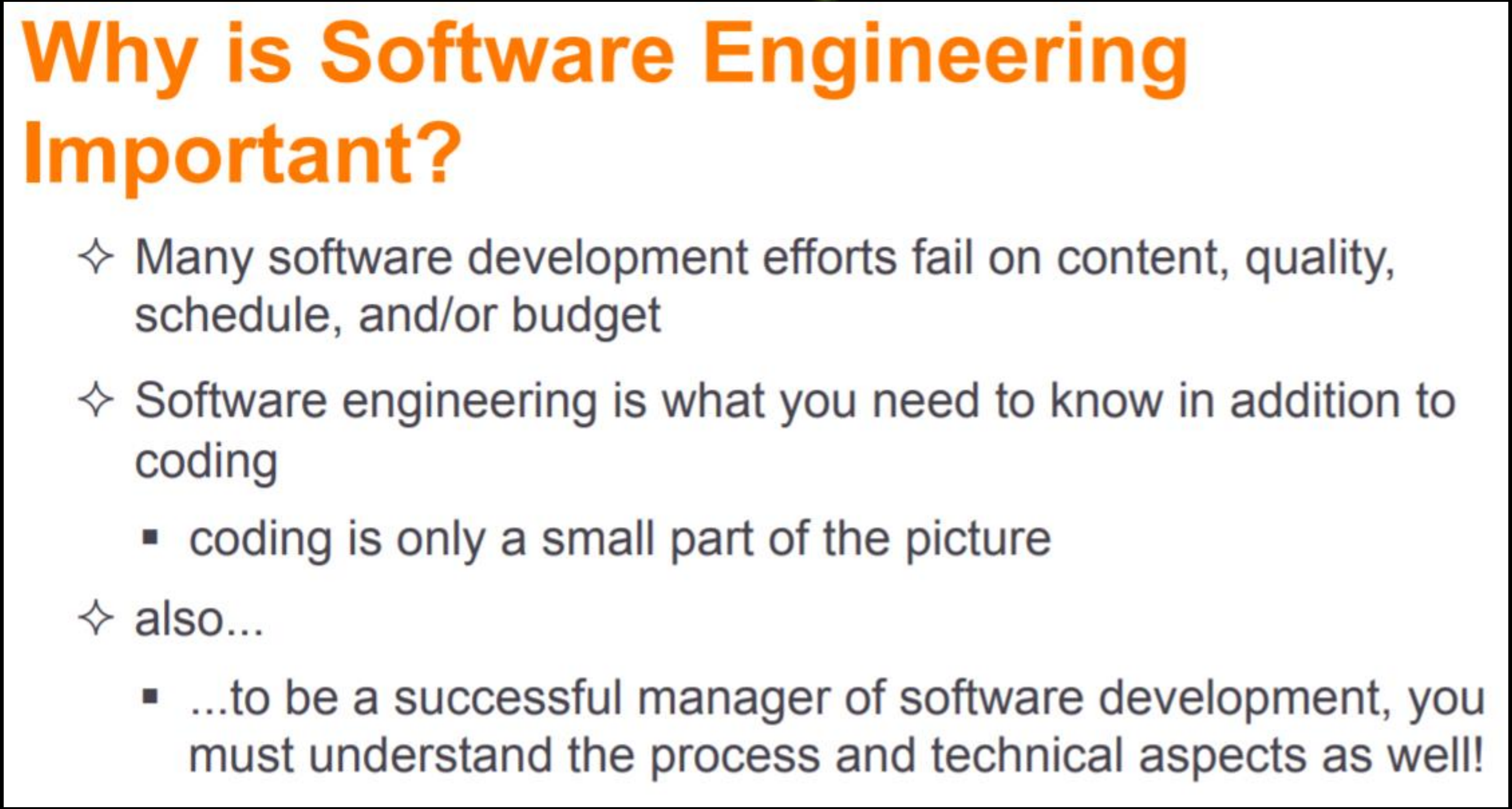
LUT University

SOME ROLES YOU CAN WORK IN AFTER THIS PROGRAM

- software developer
- test engineer
- automation engineer
- software designer
- business process analyst
- business service innovator
- digital transformation consultant

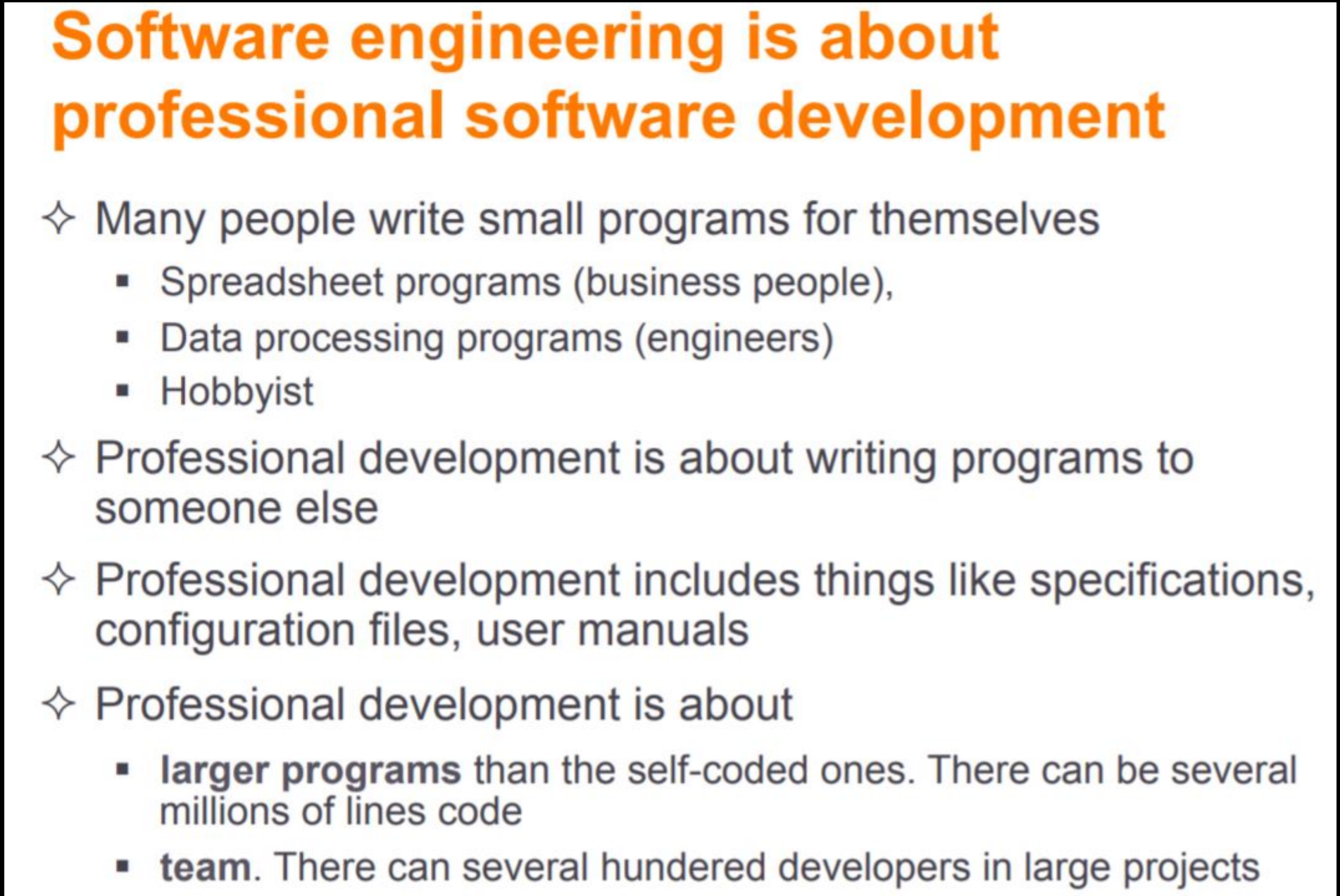
YOUR TASK NOW:

- Form 3-4 person teams
- Search for job adds for: “software engineers”
- Questions:
  - What kind of companies? (Size, industry, etc.)
  - What kind of jobs? (Roles etc.)
  - What kind of skills needed?
  - Salaries?
  - Countries?
  - Did you find your dream job? What is it?
- Search especially software engineering jobs that would be interesting to you
- Create a short presentation on your findings as a team
- Present in 2-3 min



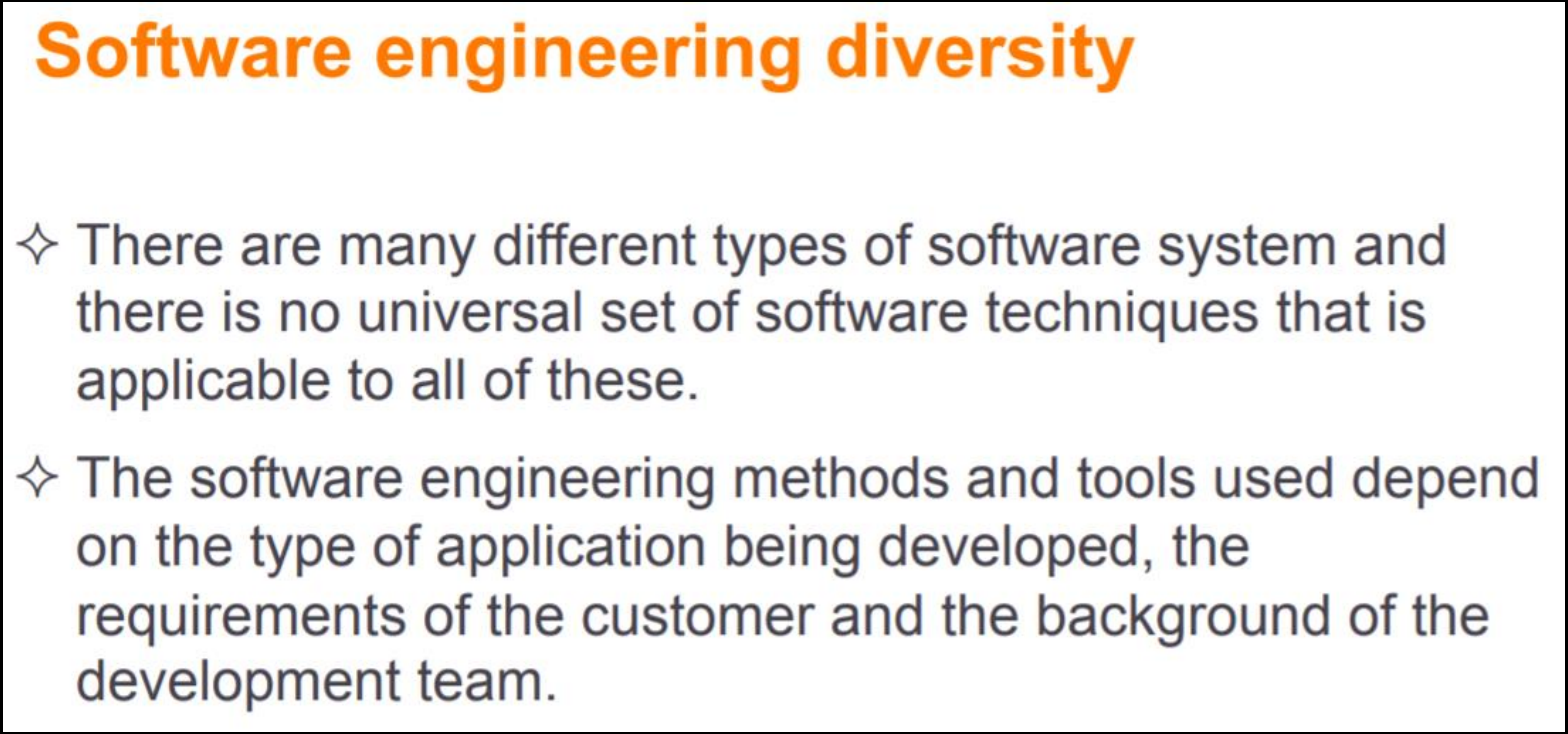
Why is Software Engineering Important?

- Many software development efforts fail on content, quality, schedule, and/or budget
- Software engineering is what you need to know in addition to coding
  - coding is only a small part of the picture
- also...
  - ...to be a successful manager of software development, you must understand the process and technical aspects as well!



Software engineering is about professional software development

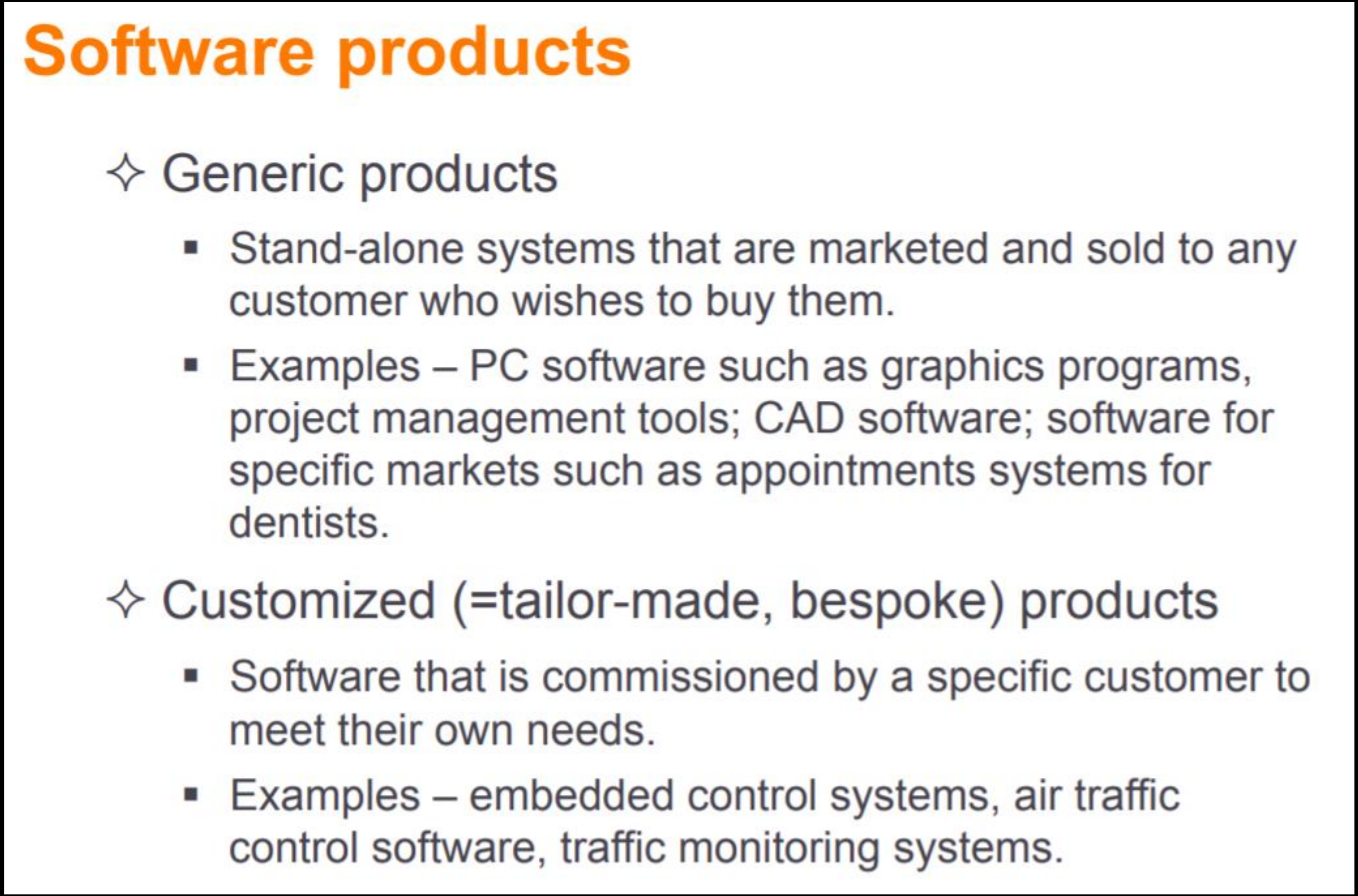
- Many people write small programs for themselves
  - Spreadsheet programs (business people),
  - Data processing programs (engineers)
  - Hobbyist
- Professional development is about writing programs to someone else
- Professional development includes things like specifications, configuration files, user manuals
- Professional development is about
  - **larger programs** than the self-coded ones. There can be several millions of lines code
  - **team**. There can several hundred developers in large projects



Software engineering diversity

- There are many different types of software system and there is no universal set of software techniques that is applicable to all of these.
- The software engineering methods and tools used depend on the type of application being developed, the requirements of the customer and the background of the development team.

LUT University

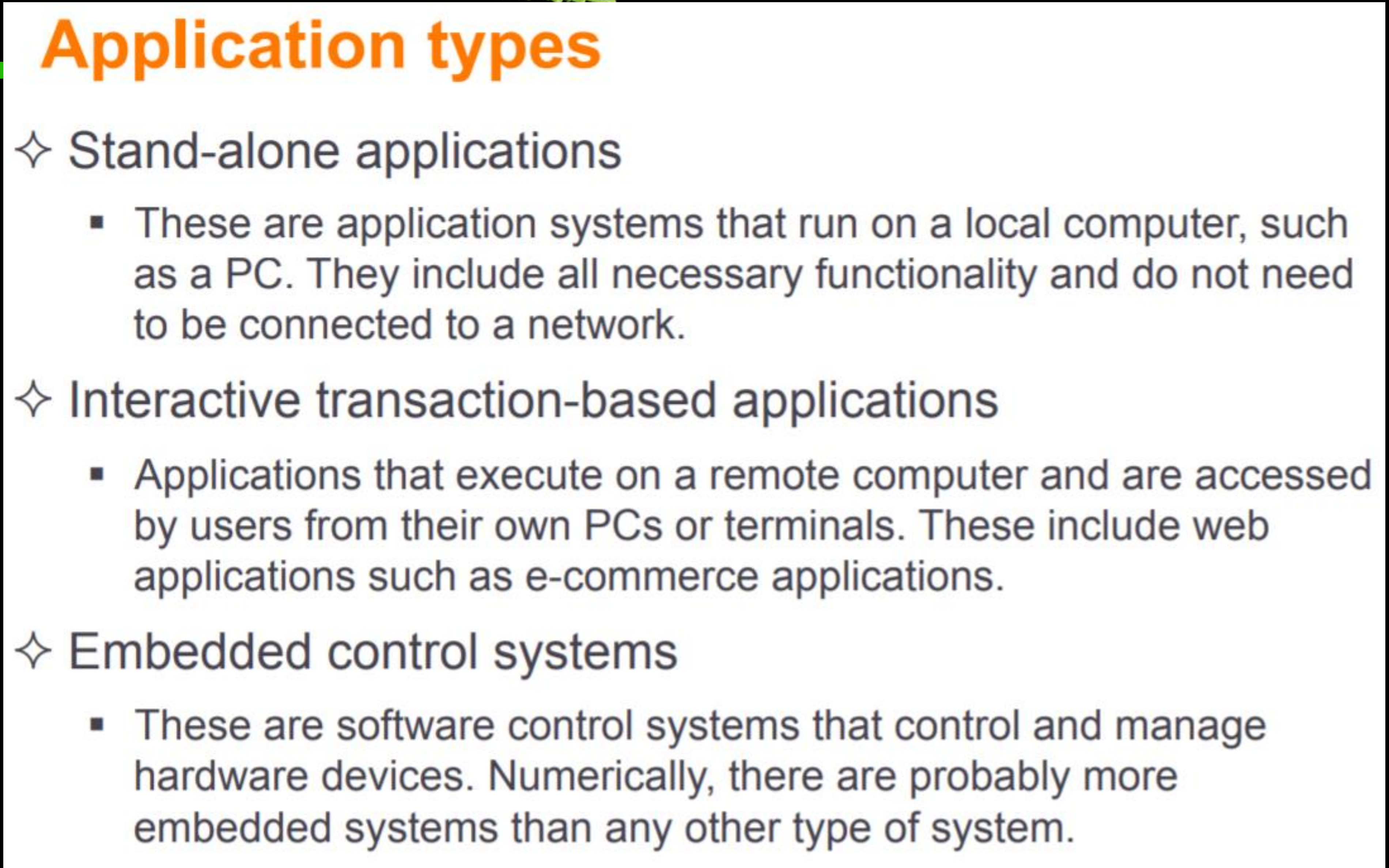


Software products

- ✧ Generic products
  - Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
  - Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
- ✧ Customized (=tailor-made, bespoke) products
  - Software that is commissioned by a specific customer to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Product specification

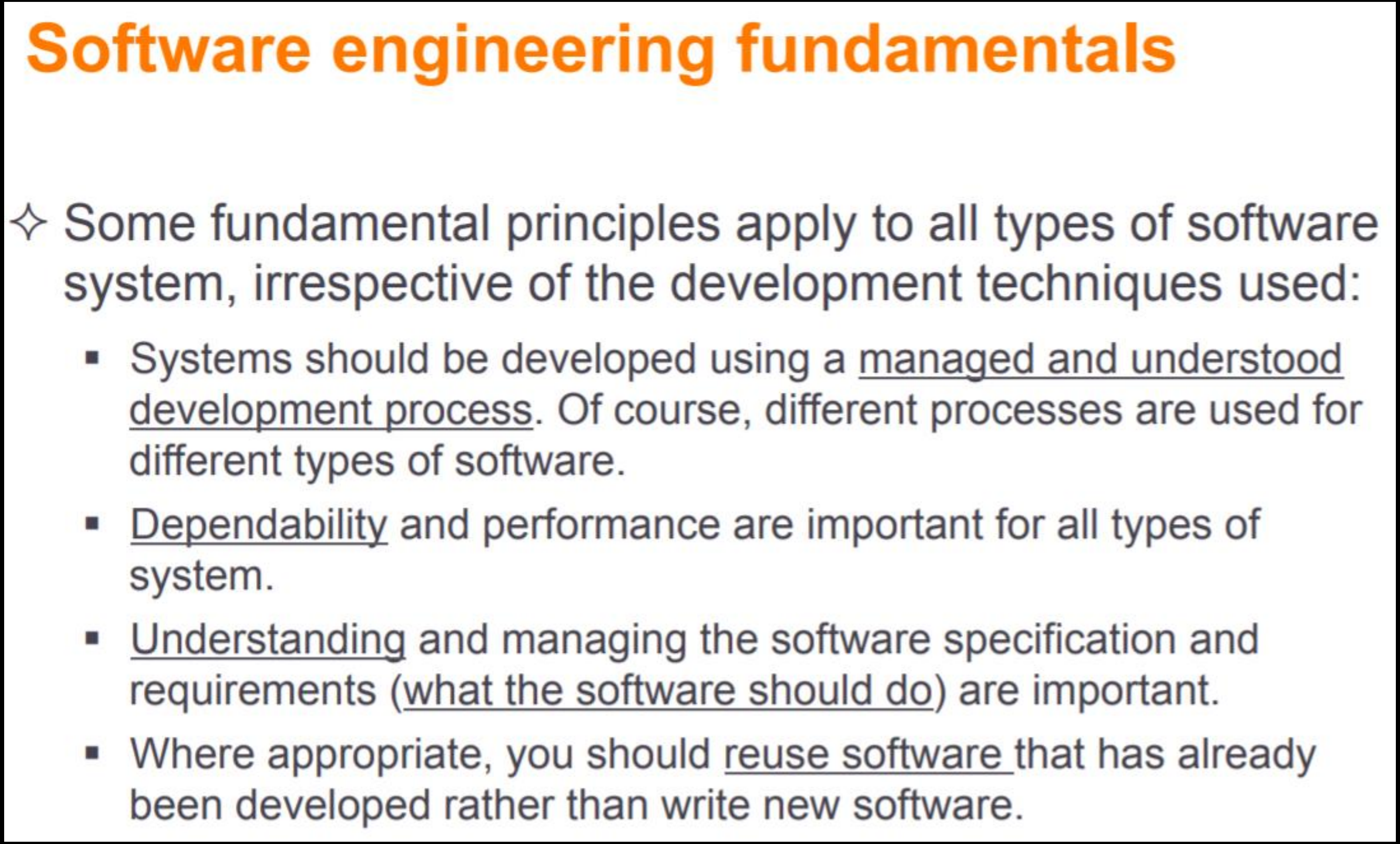
- Generic products
  - The specification of what the software should do is owned by the organization developing software and decisions on software change are made by the developing organization.
- Customized products
  - The specification of what the software should do is owned by the customer for the software and they make decisions on software changes that are required.



Application types

- ✧ Stand-alone applications
  - These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.
- ✧ Interactive transaction-based applications
  - Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
- ✧ Embedded control systems
  - These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.

LUT University

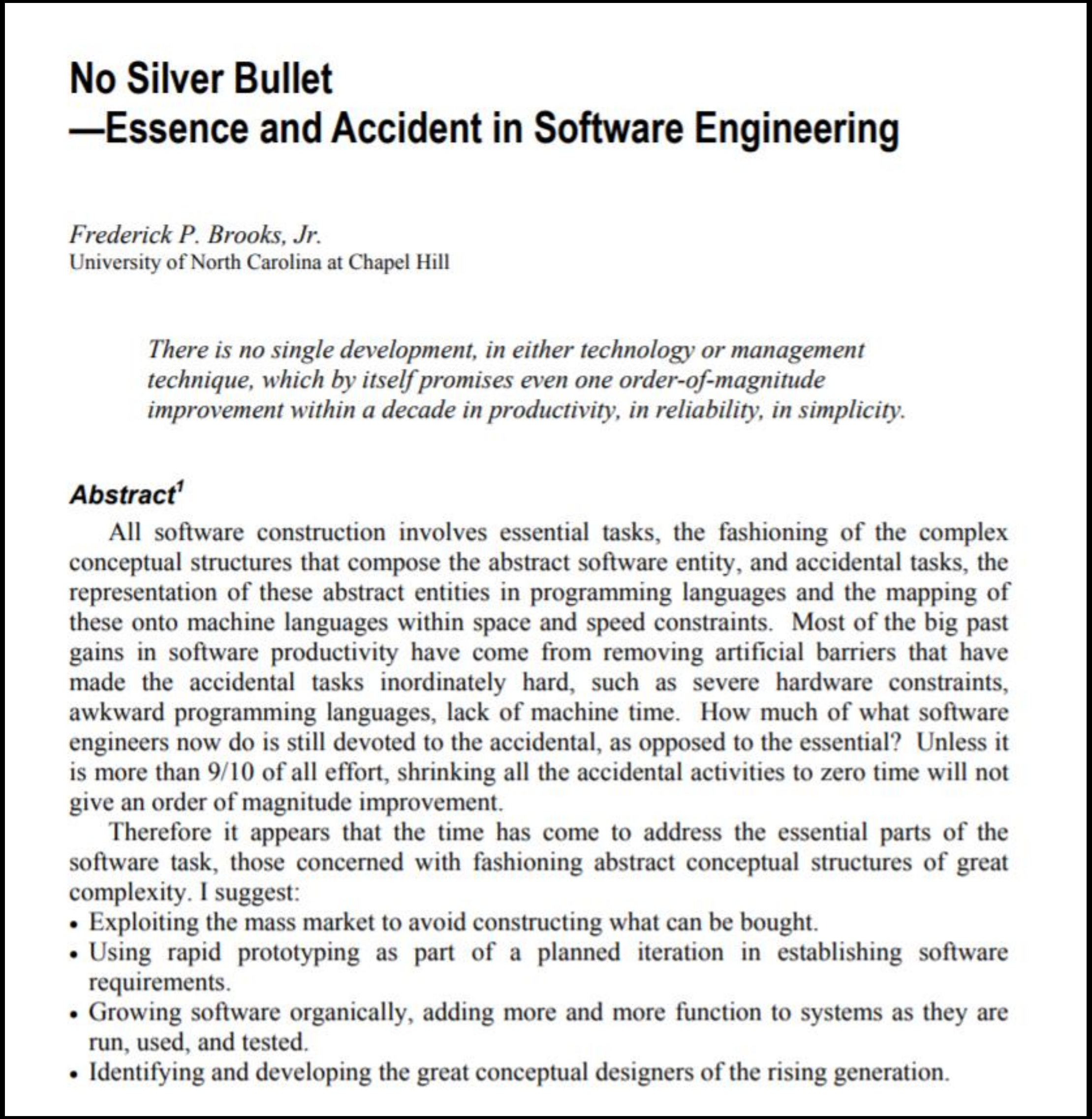


Software engineering fundamentals

- Some fundamental principles apply to all types of software system, irrespective of the development techniques used:
  - Systems should be developed using a managed and understood development process. Of course, different processes are used for different types of software.
  - Dependability and performance are important for all types of system.
  - Understanding and managing the software specification and requirements (what the software should do) are important.
  - Where appropriate, you should reuse software that has already been developed rather than write new software.

ARTICLE 1: “NO SILVER BULLET – ESSENCE AND ACCIDENT IN SOFTWARE ENGINEERING”

- By Frederick Brooks, year 1987
- How to read?
  - Try to understand the main points
  - What does “No silver bullet” mean here?
  - The inherent properties: Complexity, Conformity, Changeabilty, Invisibility
  - What are his suggestions for the future, in software engineering?
- Reference: Brooks, F. P. J. (1987) ‘No Silver Bullet Essence and Accidents of Software Engineering’, Computer 20(4), pp. 10–19. doi: 10.1109/MC.1987.1663532.



No Silver Bullet —Essence and Accident in Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

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.

Abstract[1]

All software construction involves essential tasks, the fashioning of the complex conceptual structures that compose the abstract software entity, and accidental tasks, the representation of these abstract entities in programming languages and the mapping of these onto machine languages within space and speed constraints. Most of the big past gains in software productivity have come from removing artificial barriers that have made the accidental tasks inordinately hard, such as severe hardware constraints, awkward programming languages, lack of machine time. How much of what software engineers now do is still devoted to the accidental, as opposed to the essential? Unless it is more than 9/10 of all effort, shrinking all the accidental activities to zero time will not give an order of magnitude improvement.

Therefore it appears that the time has come to address the essential parts of the software task, those concerned with fashioning abstract conceptual structures of great complexity. I suggest:

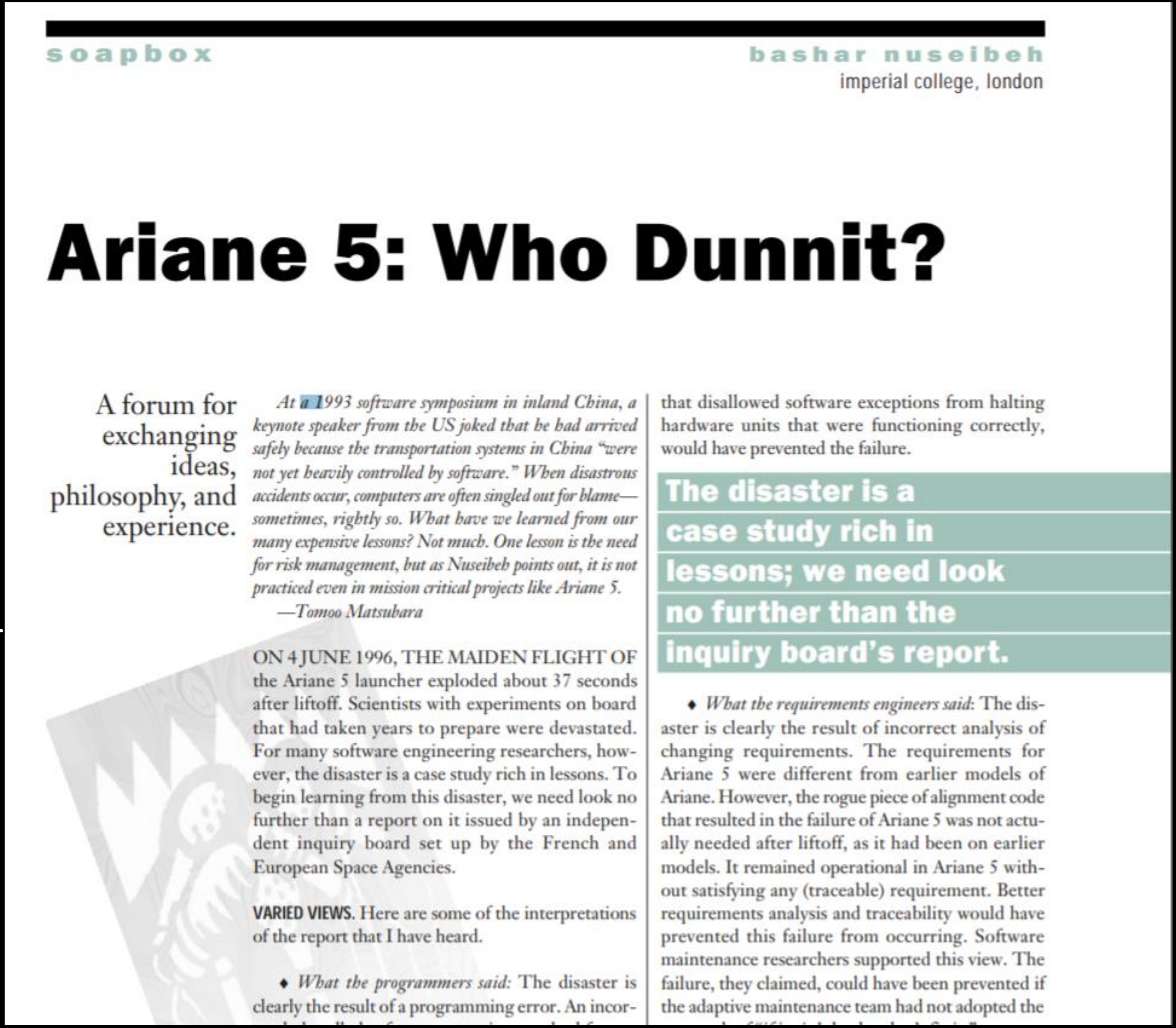
- Exploiting the mass market to avoid constructing what can be bought.
- Using rapid prototyping as part of a planned iteration in establishing software requirements.
- Growing software organically, adding more and more function to systems as they are run, used, and tested.
- Identifying and developing the great conceptual designers of the rising generation.

SOFTWARE ENGINEERING DISASTERS

- 4th June 1996. Approximately 37 seconds after a successful lift-off, the Ariane 5 launcher lost control (a 370 million dollar firework).
- The crash report identified a software bug as the direct cause (integer overflow).
- Incorrect control signals were sent to the engines and these swivelled so that unsustainable stresses were imposed on the rocket.
- It started to break up and was destroyed by ground controllers.
- The system failure was a direct result of a software failure. However, it was symptomatic of a more general systems validation failure.
- http://en.wikipedia.org/wiki/Ariane_5_Flight_501

ARTICLE 2: “ARIANE 5: WHO DUNNIT?”

- Year 1997
- How to read?
  - What can we learn from this?
- Reference: Nuseibeh, B. (1997) ‘Ariane 5: Who Dunnit?’, IEEE Software, 14(3), pp. 15–16. doi: 10.1109/MS.1997.589224.



soapbox

bashar nuseibeh
imperial college, london

Ariane 5: Who Dunnit?

A forum for exchanging ideas, philosophy, and experience.

At a 1993 software symposium in inland China, a keynote speaker from the US joked that he had arrived safely because the transportation systems in China “were not yet heavily controlled by software.” When disastrous accidents occur, computers are often singled out for blame—sometimes, rightly so. What have we learned from our many expensive lessons? Not much. One lesson is the need for risk management, but as Nuseibeh points out, it is not practiced even in mission critical projects like Ariane 5.
—Tomoo Matsubara

ON 4 JUNE 1996, THE MAIDEN FLIGHT OF the Ariane 5 launcher exploded about 37 seconds after liftoff. Scientists with experiments on board that had taken years to prepare were devastated. For many software engineering researchers, however, the disaster is a case study rich in lessons. To begin learning from this disaster, we need look no further than a report on it issued by an independent inquiry board set up by the French and European Space Agencies.

VARIED VIEWS. Here are some of the interpretations of the report that I have heard.

- *What the programmers said:* The disaster is clearly the result of a programming error. An incor-

that disallowed software exceptions from halting hardware units that were functioning correctly, would have prevented the failure.

The disaster is a case study rich in lessons; we need look no further than the inquiry board’s report.

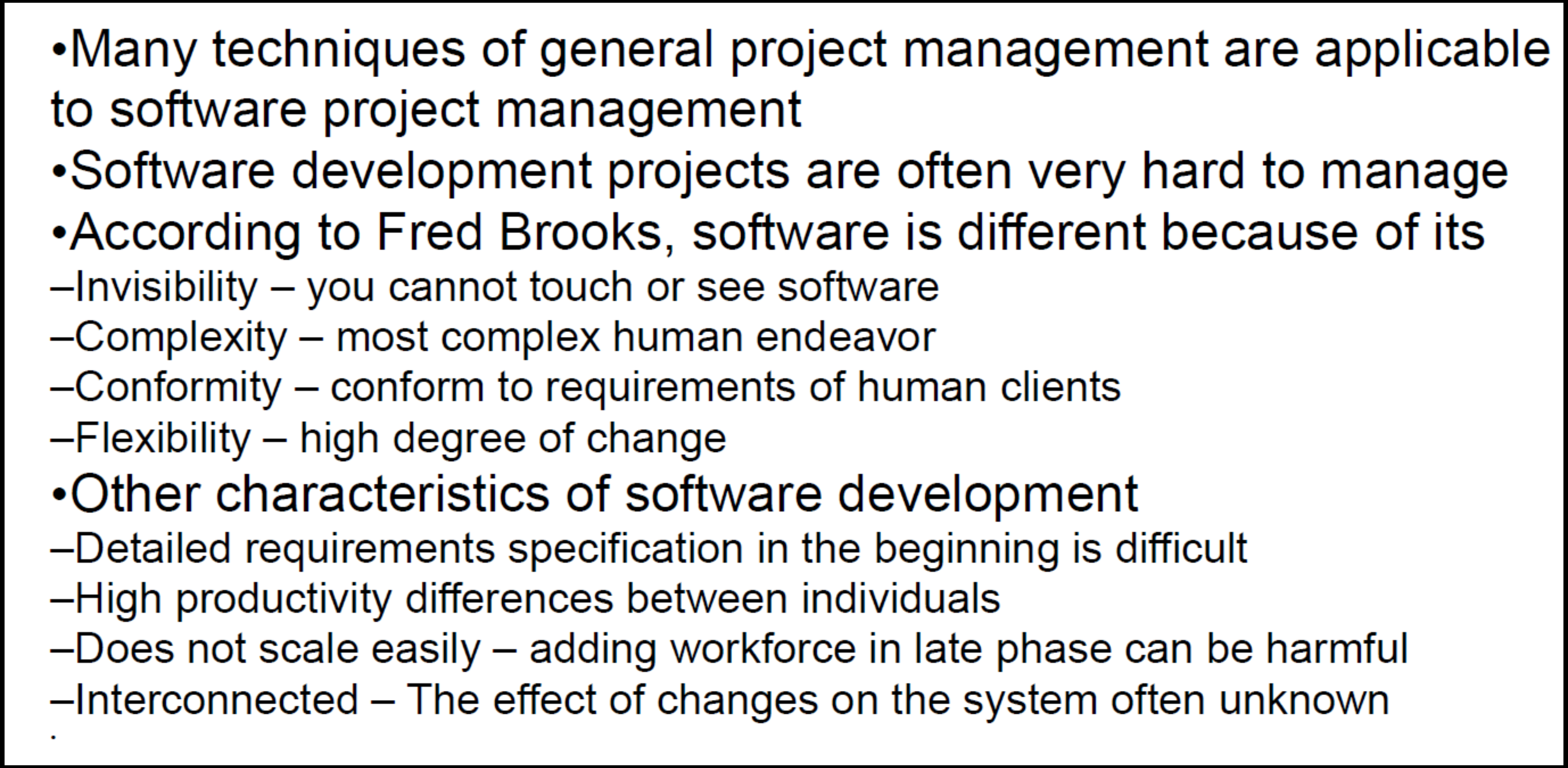
- *What the requirements engineers said:* The disaster is clearly the result of incorrect analysis of changing requirements. The requirements for Ariane 5 were different from earlier models of Ariane. However, the rogue piece of alignment code that resulted in the failure of Ariane 5 was not actually needed after liftoff, as it had been on earlier models. It remained operational in Ariane 5 without satisfying any (traceable) requirement. Better requirements analysis and traceability would have prevented this failure from occurring. Software maintenance researchers supported this view. The failure, they claimed, could have been prevented if the adaptive maintenance team had not adopted the

EXAMPLE: FINNISH RAILWAY TICKETING SYSTEM

- New system, new functionality (e.g. flexible pricing)
- 3 years of development, cost 15 mil. EUR
- 3 hours in use before failure (year 2012)
- Three times more customers during the first day than the highest expected number of customers
- System out of use for 2 weeks
- The main reason for failure: more customers than expected (1,2 milj)
- Many more reasons found: lack of testing, took into use over night everywhere, globally distributed team, several companies involved, etc.

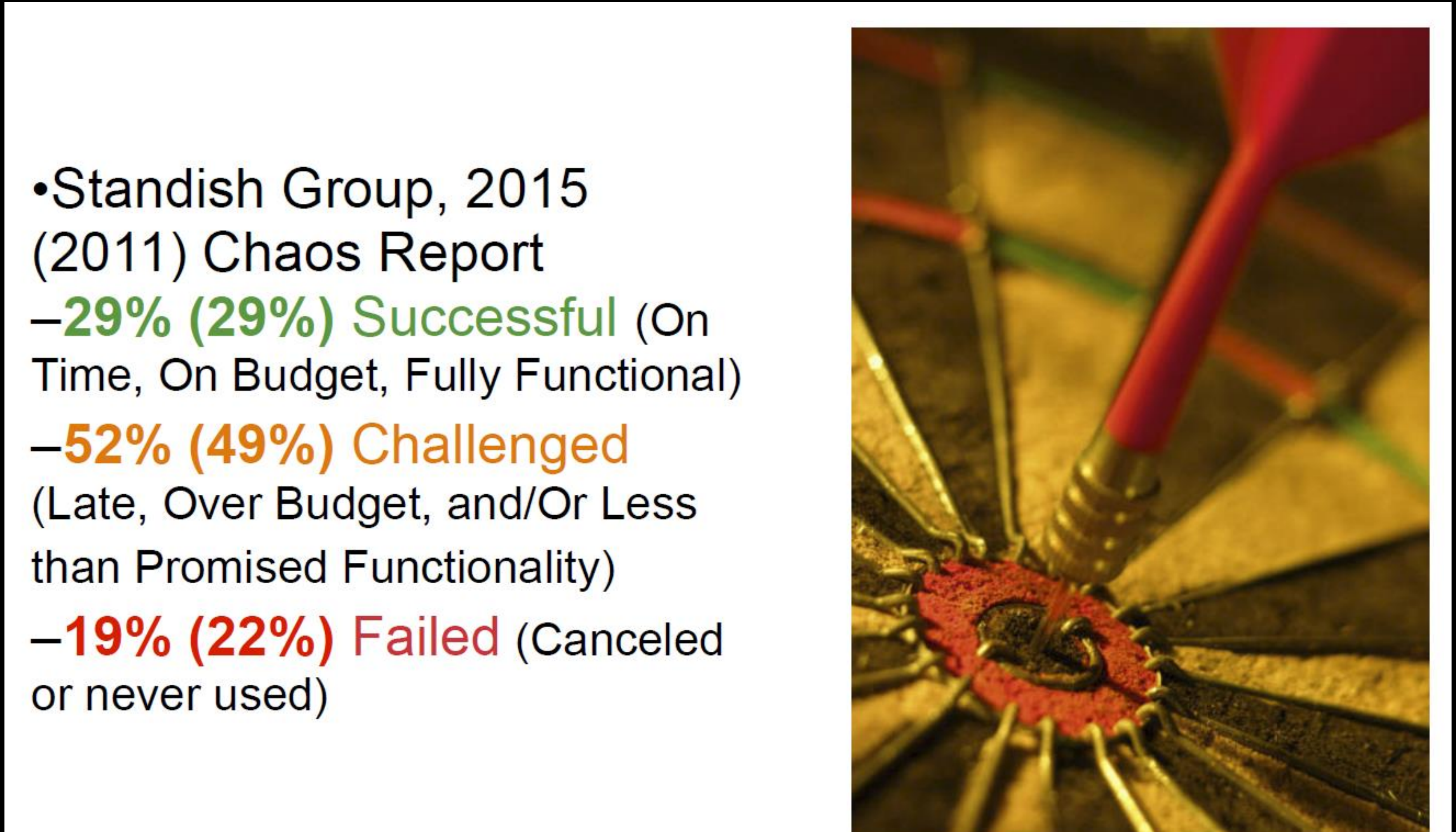
LUT University

MANAGING SOFTWARE DEVELOPMENT VS. OTHER PROJECTS



- Many techniques of general project management are applicable to software project management
- Software development projects are often very hard to manage
- According to Fred Brooks, software is different because of its
  - Invisibility – you cannot touch or see software
  - Complexity – most complex human endeavor
  - Conformity – conform to requirements of human clients
  - Flexibility – high degree of change
- Other characteristics of software development
  - Detailed requirements specification in the beginning is difficult
  - High productivity differences between individuals
  - Does not scale easily – adding workforce in late phase can be harmful
  - Interconnected – The effect of changes on the system often unknown

SOFTWARE PROJECT SUCCESS RATES

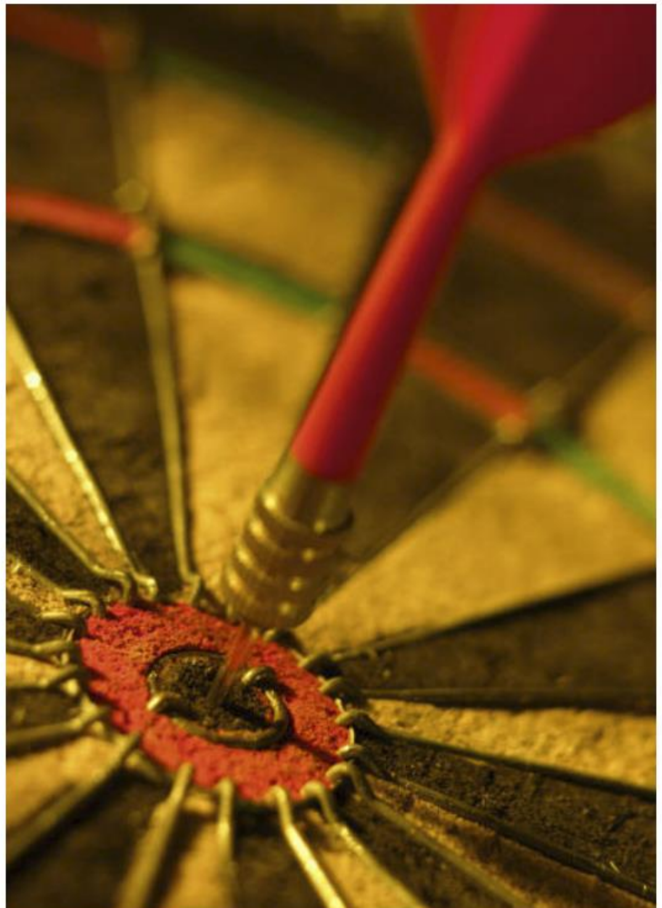


•Standish Group, 2015 (2011) Chaos Report

–**29% (29%)** Successful (On Time, On Budget, Fully Functional)

–**52% (49%)** Challenged (Late, Over Budget, and/Or Less than Promised Functionality)

–**19% (22%)** Failed (Canceled or never used)

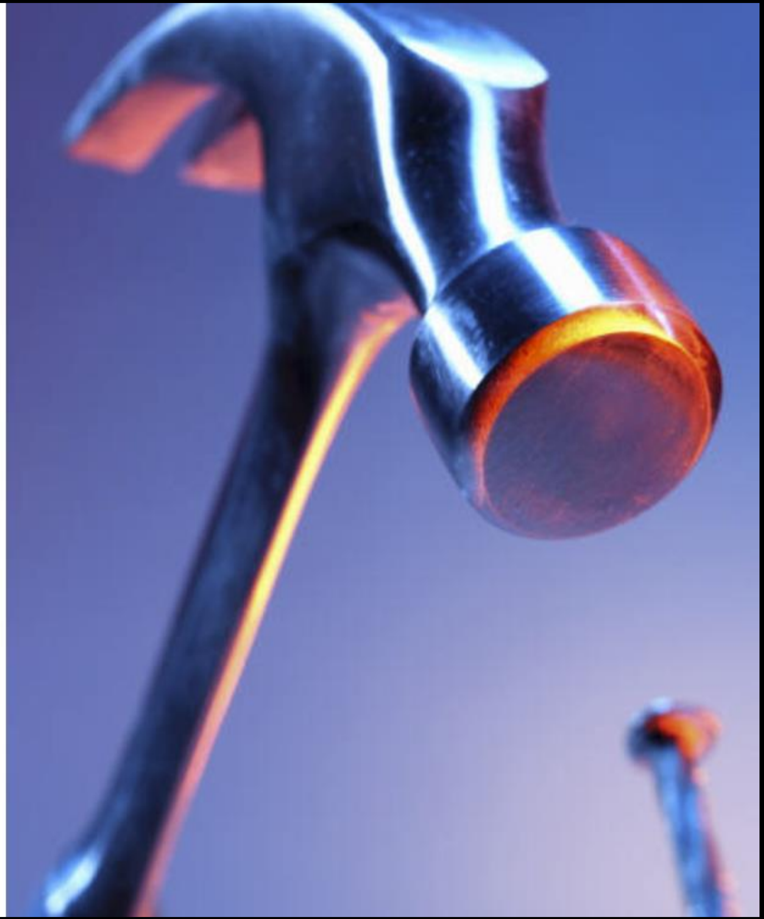


REASONS FOR FAILURE (ACCORDING TO STANDISH GROUP)

- "Most projects failed for lack of skilled project management and executive support"
- "Underestimating project complexity and ignoring changing requirements are basic reasons why projects fail"
- "The problem – and the solution – lay in people and processes"

RECIPE FOR SUCCESS

- Smaller project size and shorter duration
- More manageable
- “Growing”, instead of “developing”, software engages the users earlier and confers ownership.
- -> Iterative and interactive process

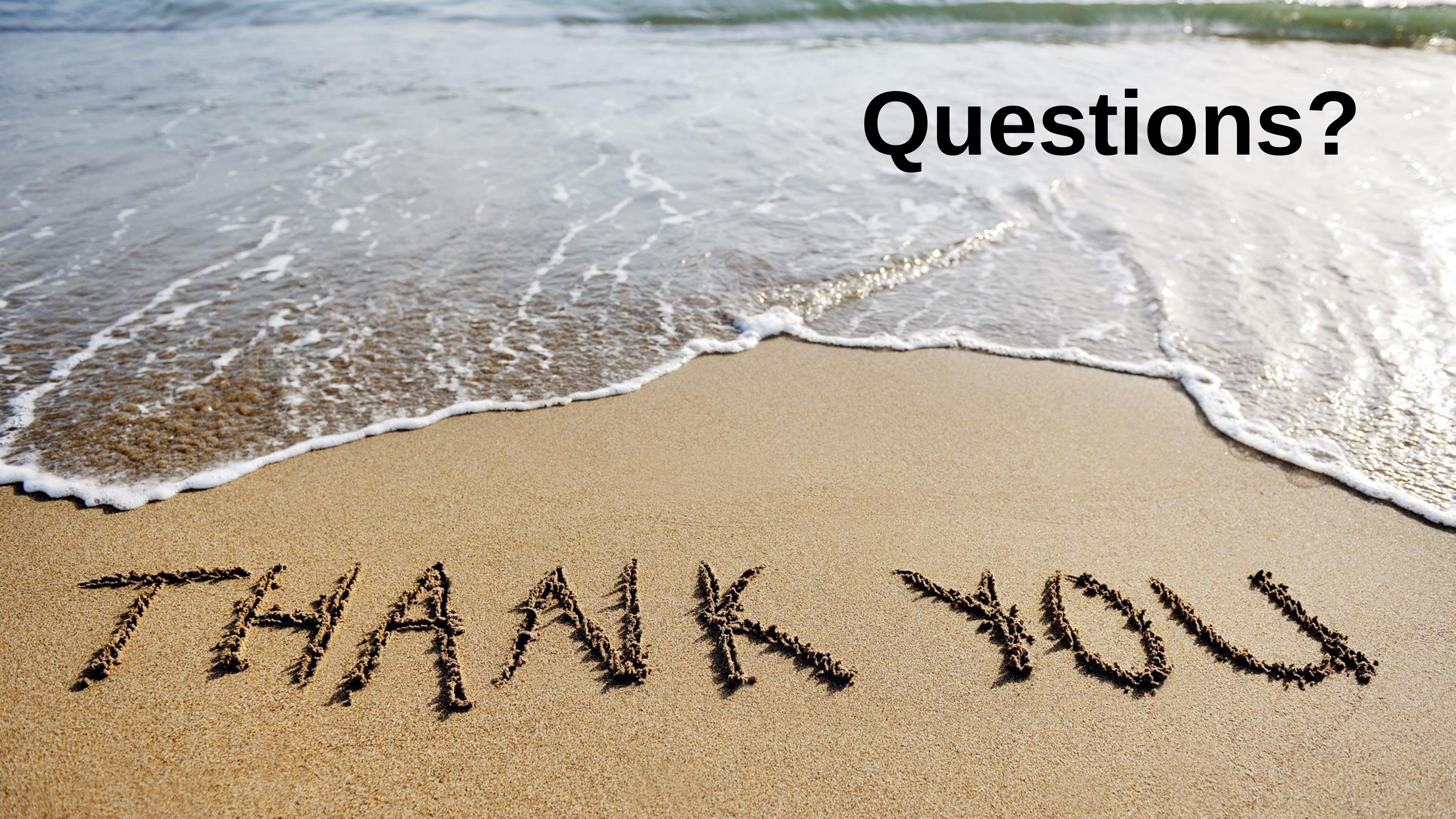


ESSAY 1: SOFTWARE ENGINEERING DISASTERS

- Research software engineering disasters and their costs. (Tips: google for "software engineering disasters"). **Select two disasters** for which you can find good descriptions, and that somehow are interesting to you.
- Write an essay describing at least two of the disasters, the reasons behind them, as well as their direct and indirect costs. Do not forget to reference according to the instructions.
- What do you think could be done to prevent software engineering disasters in the future?
- Finally, **describe what your learning goals** are for this course in software engineering. List at least three goals and explain why they are meaningful to you.
- Aim for a total length of about 1000 words. If your essay is significantly shorter (less than 750 words) or longer (more than 1250 words), you will be penalized in the grading for not adhering to the length requirements.

ESSAY WILL BE GRADED ON:

- the quality and understanding shown in the descriptions of the software engineering disasters and your thoughts on how to prevent such disasters in the future.
- the existence of motivated learning goals
- the length, as described above
- the correct use of referencing. Note that absense of references fails the essay completely, as you are required to reference)
- an overall assessment of the essay quality



Questions?

LUT University