Introduction to Software Engineering

Prof. Walter F. Tichy

ქუთაისის
საერთაშორისო
უნივერსიტეტი

Introduction

Def: **Software Engineering** is the technological and managerial discipline concerned with the systematic development and maintenance of software and software-intensive systems satisfying specified functional and quality attributes.

Introduction

Def: **Software** consists of programs, associated data, and documentation for performing tasks with the aid of computers.

Excluded from this definition is pure data, i.e., books, films, music, static web pages, databases, etc., without any executable programs.

Introduction

- Examples of programs: source code, object code, intermediate code, libraries, frameworks, installation programs, test programs.

- Examples of associated data: initialization data, examples, user manuals, help information, error messages, architectural description, program documentation, test data, expected test output.

Introduction

Managerial Aspects of SE:

- **Planning:** who does what, when, where, and how; cost estimation.
- **Staffing:** finding appropriate personnel.
- **Organizing:** establishing framework and rules for working together (tools, processes, tasks, responsibilities, reporting)
- **Directing:** motivating workers and communicating goals
- **Control:** assuring work progresses according to plan.

In this course, we cover of the managerial tasks only **planning** and **organizing**. The emphasis will be on technical aspects.

Introduction

Technical Aspects of SE:

- Methods for software development and maintenance
  (e.g., object-oriented analysis and design, inspections, testing methods)
- Tools that automate or simplify steps in SW development and maintenance
  (e.g., languages and compilers, program generators, design tools, test tools, configuration management tools, clone detectors, programming environments such as Eclipse or Studio)

Introduction

- **Systematic:** “systematic” means one has defined processes, methods and tools; workers are not guessing or trying out ideas.

- **Development and maintenance:** successful software is changed after its initial development, because corrections, adaptations, and extensions become necessary.

- Maintenance can amount to 60% and more of lifecycle costs.

Introduction

Specified functional and quality attributes:

- **Functional attributes** specify what the software does, i.e., its function.
- **Quality attributes** (also called **non-functional attributes**) specify how well the software performs its function (how reliably, how quickly, how user-friendly) and other properties such as security and modifiability.

Introduction

- Observation:
- Software is known to be difficult to develop on time, within budget, and of satisfactory quality.

Introduction

Example 1: **Automatic Baggage System of the Denver Airport.**

- Gigantic airport, twice the area of Manhatten,
  10 times the width of Heathrow.
  3 airplanes can land simultaneously, in bad weather.
- Automatic baggage system:
- 34 km long
- 4000 automatic vehicles
- 100 control computers
- 5000 optical sensors
- 400 radios
- 56 barcode readers

Introduction

Denver Airport

Baggage System of Denver Airport

Introdcution

Denver Airport continued:

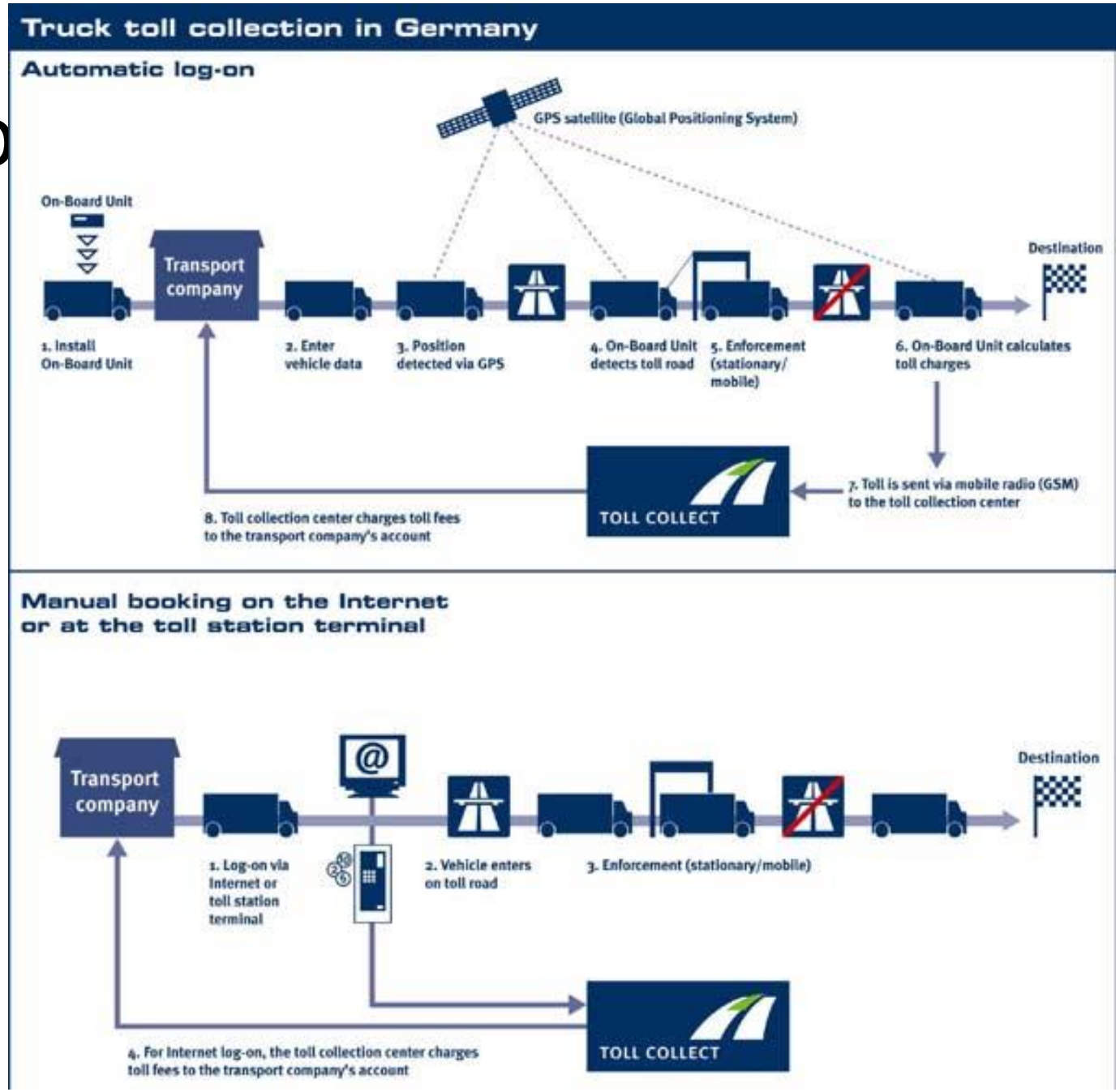
- Planned opening: 31 Oct. 1993
  - Baggage system delayed due to software problems
  - First delay to Dec. 1993, then March 1994.
  - June 1994: no estimate for completion
- Final start of operation: Feb. 1995 (16 months late)
- Financial loss: 1.1 Million $ / day (interest, operations)
- The baggage system was scrapped in 2005 and replaced with a traditional system with conveyor belts and trolleys. The automated system never worked properly.

Introduction

Example 2: Germany's **TollCollect**

- TollCollect is a system for automatic collection of tolls for heavy trucks on German autobahns.
- Novel aspects: Trucks do not stop at toll stations; there are *no* toll collection stations!
- Instead, each truck has an onboard computer with a GPS sensor that tracks the route of the truck, automatically computes the toll, and sends the information via mobile telephony (SMS!) to a central billing location. This location issues invoices and debits credit cards.
- (Trucks without onboard computers can prepay their tolls at web terminals.)

Intro


Truck toll collection in Germany
Automatic log-on
GPS satellite (Global Positioning System)
On-Board Unit
Transport company
Destination
1. Install On-Board Unit
2. Enter vehicle data
3. Position detected via GPS
4. On-Board Unit detects toll road
5. Enforcement (stationary/ mobile)
6. On-Board Unit calculates toll charges
7. Toll is sent via mobile radio (GSM) to the toll collection center
8. Toll collection center charges toll fees to the transport company's account
TOLL COLLECT
Manual booking on the Internet or at the toll station terminal
Transport company
@
Destination
1. Log-on via Internet or toll station terminal
2. Vehicle enters on toll road
3. Enforcement (stationary/mobile)
4. For Internet log-on, the toll collection center charges toll fees to the transport company's account
TOLL COLLECT

Introduction

TollCollect continued:

- Planned start of operation: 31 August 2003
- Software problems delayed the start (onboard units crashed; some could not be restarted; central system was overloaded with requests for updates).
- Alternative, “Eurovignette” (a sticker) was cancelled.
- Reduced system operational January 2005 (without radio updates of road maps and tariffs).
- Full system operational 1 January 2006.
- Total loss: 5600 Million Euro plus interest.

Introduction

- Why are there these spectacular software project failures?
- Software is immaterial:
  cannot be touched, has no physical representation, state of development is difficult to determine.

- Software is difficult to measure:
  What can be measured (lines of code and other metrics) has little to do with software function, quality, development state.

Introduction

- Software is not subject to physical laws and limits:
  - the scope and scale of a software design has no observable limit
  - because of software's discrete (non-continuous) nature, small faults can have dramatic effects. For example, a loop index that is off by 1 can cause a whole program to crash.

Introduction

- Software doesn't wear out and there are no spare parts:
  - Software is either correct or incorrect from the beginning
- Software does seem to age:
  - The environment around software changes so quickly and dramatically that software that isn't adapted to these changes becomes unusable quickly
- Examples:
  - a database system without a web interface is hardly useful since the arrival of the World Wide Web.
  - Software must be adapted to smart phones, tablets
  - Operating systems are replaced every 4-5 years, meaning that existing software needs to be ported to the new OS or become obsolete.

Introduction

- Software can be extremely complex:

  Operating systems or telecommunication systems are probably more complex than any physical system created by man, and beyond the grasp of a single individual. (example: Windows XP was 35 Million lines of code)
- Software has a longer life span than the hardware it runs on: Hardware platforms are often replaced after 3-5 years; some software systems are in use 10 or 20 years or longer. (Therefore: When building software, plan for replacing hardware and operating system twice.)

Introduction

- To master these difficulties, a systematic approach to development and maintenance of software is required.
- Otherwise, the software becomes too expensive, cannot be finished on time, is unreliable, or not maintainable.
- There is one tremendous advantage of software:
  It can be replicated and distributed nearly for free. There is no manufacturing process as for physical products, such as cars or buildings. No raw materials must be bought. Once the software is implemented (or corrected or improved or extended), it can be replicated, distributed, and replaced millionfold at negligible cost.

Introduction

- Ubiquity of Software
- Software is everywhere in our modern lives.
- Software embedded in devices has become the dominant value enhancer.
- Name examples where you depend on software in your daily life.