COMP 433 Software Engineering

Module 1: *Introduction To Software Engineering*

Ahmed Tamrawi

Acknowledgment Notice

Part of the slides are based on content from Ian Somerville's Software Engineering book.


Software Everywhere

Software Value

- It is the fuel of modern life:
  - Business
  - Government
  - Scientists
  - Industries
  - Education
- However, **building** and **maintaining** software is *hard* and is *getting harder.*

Software Complexity

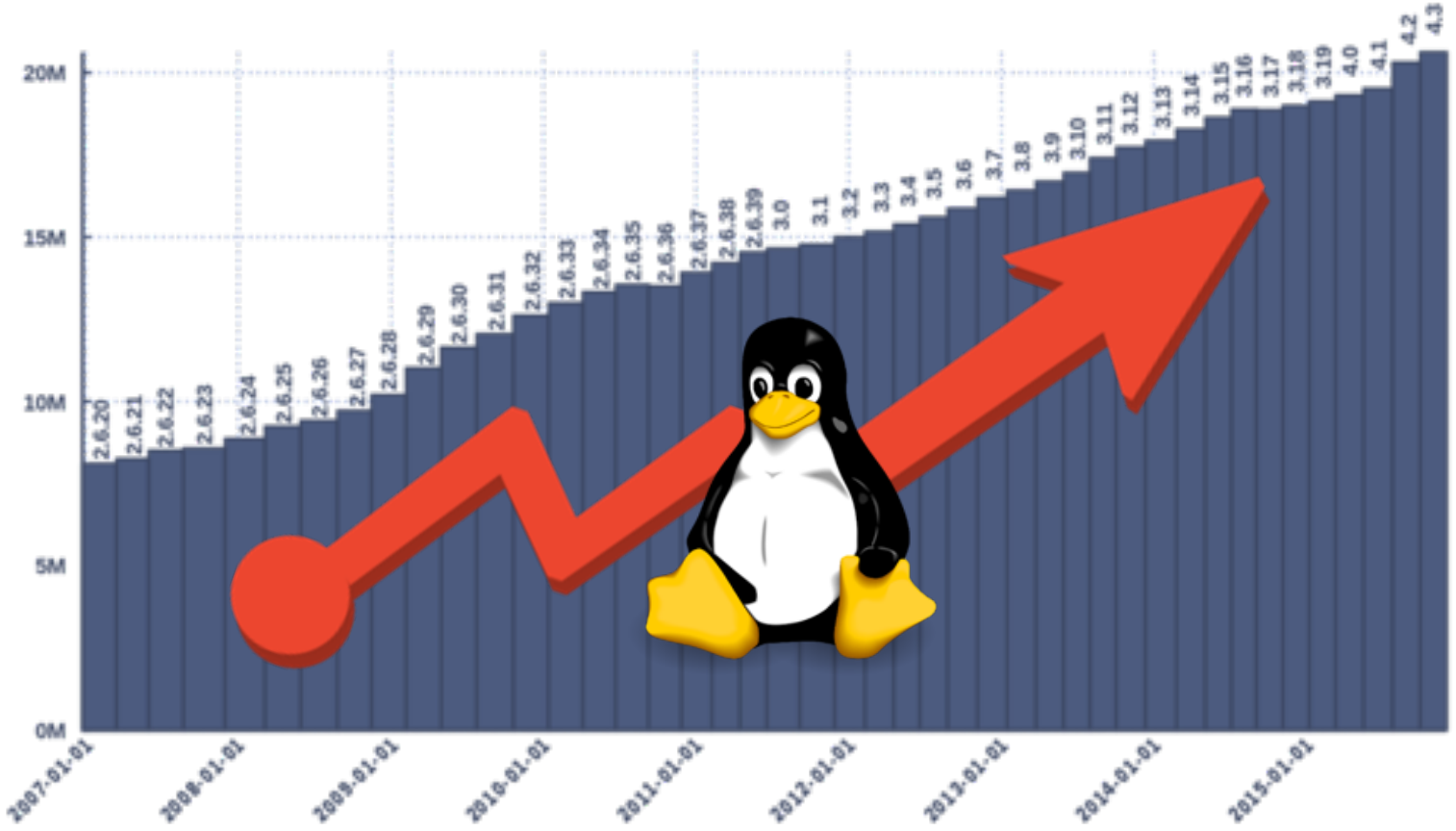
- Software development is **Complex**!
- It is important to distinguish between “**small**” systems (*one developer, one user, experimental use only*) and “**Complex**” systems (*multiple developers, multiple users, products*).
- Experience with “**small**” systems is misleading as one person techniques do not scale up.

Software Complexity

- Analogy with bridge building:
  - Building a bridge over a stream is an easy and one person job.
  - Building a bridge over a river? (*the one-person job techniques do not scale*)

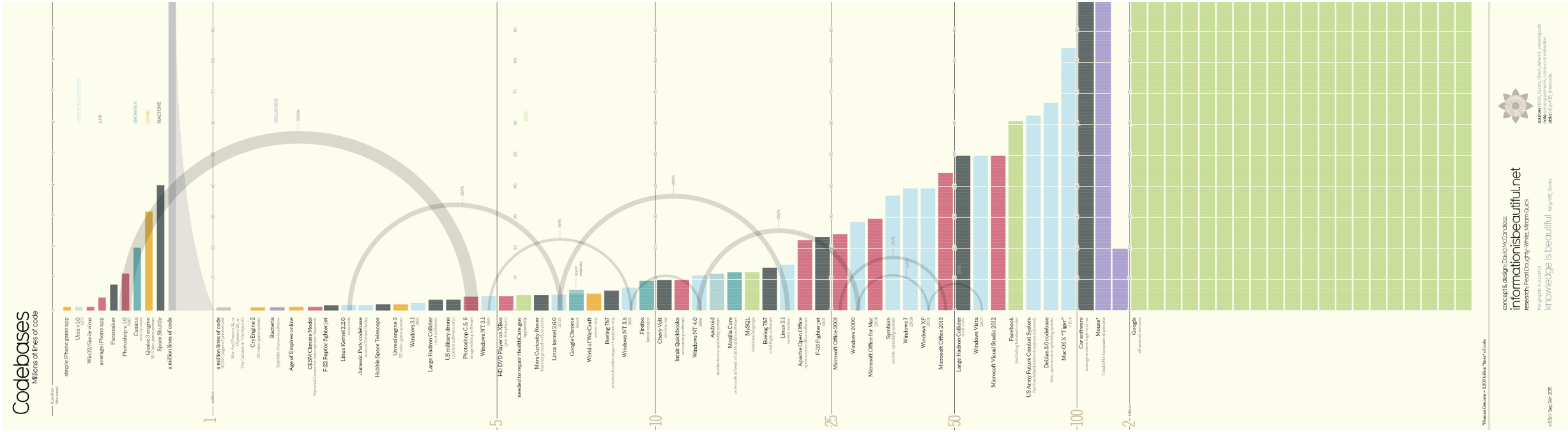
Ever Increasing Complexity

Ever *Increasing* Size



20 MLOC ≈ 360K Pages

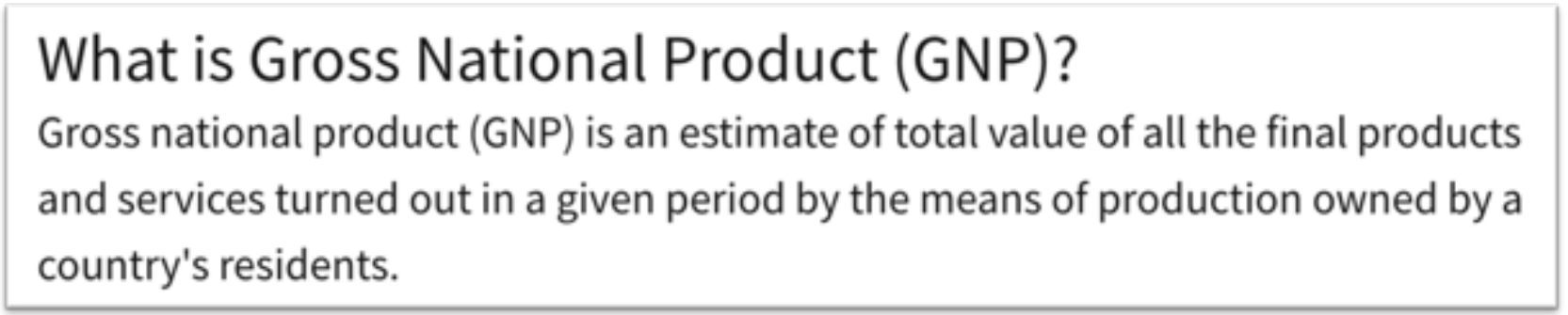
Complexity and Size *matters*!



https://www.visualcapitalist.com/millions-lines-of-code/

Software Engineering

- 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 Gross National Product (GNP) in all developed countries.



Software Costs

- Software costs often **dominate** computer system costs.
- 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 engineering is concerned with **cost-effective software development.**

Software Project Failure

- **Increasing system complexity**
  - As new software engineering techniques help us to build larger, more complex systems, the demands change. Systems have to be built and delivered more quickly; larger, even more complex systems are required; systems have to have new capabilities that were previously thought to be impossible.
- **Failure to use software engineering methods**
  - It is fairly easy to write computer programs without using software engineering methods and techniques.
  - Many companies have drifted into software development as their products and services have evolved. They do not use software engineering methods in their everyday work. Consequently, their software is often more expensive and less reliable than it should be.

Why Software Engineering?

- The problem is **complexity!**
- Complexity depends on many factors, but **size** is one key:
- For example:
  - UNIX:
    - 1971: 10,000 lines of code.
    - 2020: 27.8 millions lines of code.
  - Windows:
    - Windows 95 contains 15 millions lines of code.
    - Windows 2000 contains 100 millions lines of code.
    - Windows 7 contains 39.3 millions lines of code.
    - Windows 10 contains 50 millions lines of code.

Software Engineering is about using ***engineering methods*** to manage software complexity efficiently

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.*
- **Engineering discipline**
  - Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.
- **All aspects of software production**
  - Not just technical process of development. Also project management and the development of tools, methods etc. to support software production.

Importance of Software Engineering

- *More and more, individuals and society rely on advanced software systems*. We need to be able to produce reliable and trustworthy systems economically and quickly.
- *It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems* rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

Computer Science vs Software Engineering

- Computer science is concerned with **theory** and **fundamentals** such as: algorithms, date structures, complexity theory, numerical methods, etc.
- Software engineering is concerned with the **practicalities** of **developing** and **delivering** useful software. It deals with **practical problems** in **complex** software products.
- *Computer science theories* are currently **insufficient** to act as a complete underpinning for software engineering, but it is a foundation for *practical aspects of software engineering*.

Different than Regular Engineers?

- Regular engineers for buildings and bridges for example, have faced similar examples, risks, solutions. But as a software engineers, we are not so lucky!
- What is the real problem?
  - Complexity and Change.

Software Engineering History

- Software Engineering introduced first in 1968 – conference about "*software crisis*" when the introduction of third generation computer hardware led to **more complex** software systems than before.
- Early approaches based on *informal methodologies* led to:
  - Delays in software delivery.
  - Higher costs than initially estimated.
  - Unreliable and difficult to maintain software .
- Thus, there was an urgent need for **new methods and techniques** to *manage* the production of **complex** software.

FAQs About Software Engineering

Question	Answer
What is software?	**Computer programs and associated documentation**. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required **functionality** and **performance** to the user and should be **maintainable**, **dependable** and **usable**.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software **specification**, software **development**, software **validation** and software **evolution**.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

FAQs About Software Engineering

Question	Answer
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, **different techniques are appropriate for different types of system**. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.
What differences has the web made to software engineering?	The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

Professional Software Development

- Most of the software development is a **professional activity**:
  - Special team of developers.
  - Specific business purpose.
  - Formal and systematic processes.
  - Software will be maintained and changed throughout its life.
- Software Engineering supports **professional** software development, not *amateur* development.

Software Products

- Software engineers are concerned with developing software products, that is, software that can be sold to a customer. There are two kinds of software product:
- **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.
  - The specification of what the software should do is *owned by the software developer* and *decisions on software change* are made by the **developer**.
- **Customized 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.
  - 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.

Attributes of a Good Software

- Good software means that:
  - It *delivers the* ***required functionality***. (Functional Requirements)
  - It has ***acceptable performance*** to the user, and it should be ***maintainable***, ***dependable***, ***usable***, and ***secure***. (Non-Functional Requirements)
- Requirements specify a set of features that the system must have.
  - **Functional requirement** is a specification of a function that the system must support,
  - **Non-functional requirement** is a constraint on the operation of the system that is not related directly to a function of the system.

Attributes of Good Software

- When we talk about the **quality of professional software**, we have to consider that the software is used and changed by people apart from its developers.
- **Quality** is therefore not just concerned with what the software does. Rather, it has to include the software's behavior while it is executing and the structure and organization of the system programs and associated documentation.
- This is reflected in the software's quality or non-functional attributes.
- **Examples of these attributes** are the software's response time to a user query and the understandability of the program code.
- The specific set of attributes that you might expect from a software system obviously *depends on its application*. Therefore, an aircraft control system must be safe, an interactive game must be responsive, a telephone switching system must be reliable, and so on.

Essential Attributes of Good Software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can ***evolve to meet the changing needs of customers***. This is a critical attribute because software change is an **inevitable** requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including **reliability**, **security** and **safety**. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be ***acceptable to the type of users for which it is designed***. This means that it must be understandable, usable and compatible with other systems that they use.

Classify Requirements

1. The user must be able to purchase tickets.
2. The user must be able to access traffic information.
3. The system must be provided feedback in less than one second.
4. The colors used in the interface should be consistent with the company logo.
5. System should be easy to use since users could be of different ages.
6. Specific hardware platform for the system.
7. Strict security requirements.
8. How the system should deal with failures and faults.
9. How to provide backward compatibility with an old system that the client is unwilling to retire.

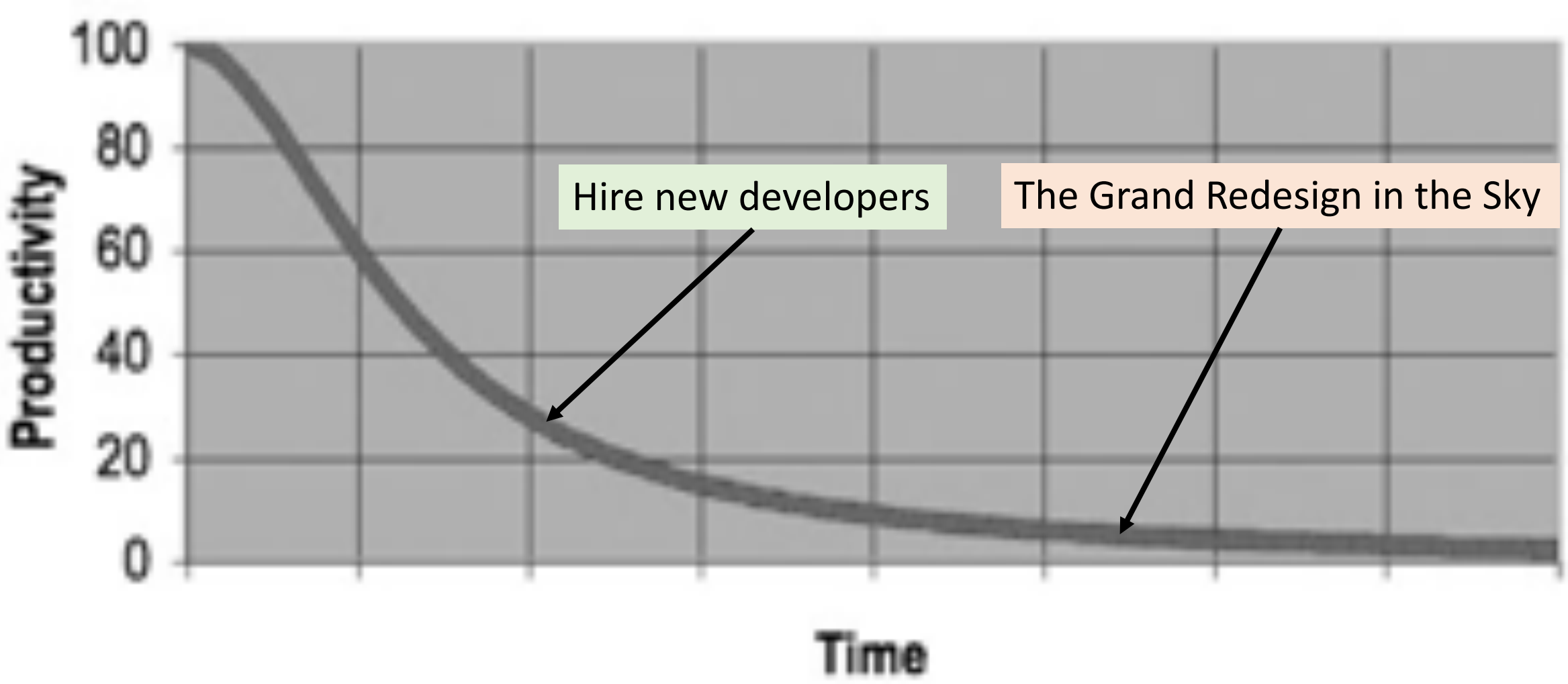
In-class Activity I

Identifying Software Myths

Activity: Software Myths

- Management myths:
  - There exist precise standards and procedures for building successful software.
  - Add more programmers if behind schedule (*see next slide*)
- Customer myths
  - General and/or generic description of objectives is enough to start coding.
  - We can change the requirements anytime as the software is flexible and easy to adapt.
- Practitioner myths
  - Task accomplished when the program works.
  - “Working program” the only project deliverable.

The Total Cost of Owning a Mess



Ariane 5 (1996)
THE 7 BILLION DOLLAR OVERFLOW

Zero-Day Flaw Linux
Taking control and privacy

NASA - Mariner 1
$18 million

Android Lollipop
https://threatpost.com/google-aware-of-memory-leakage-issue-in-android-5-1-fix-forthcoming/111640/

Car Recalls - $3
Billion

Knight Capital Trading
$440 million

July 21, 2015

Jeep remotely hijacked

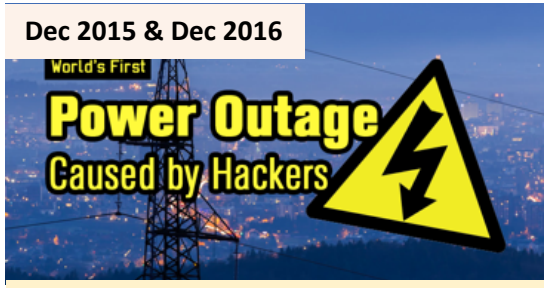
November 29, 2011

HP printers remotely set on fire

August 17, 2009

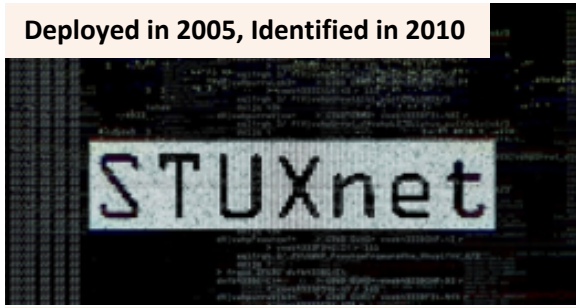
Destruction Sayano-Shushenskaya Hydroelectric Power Plant

Dec 2015 & Dec 2016



Ukraine power grid attacks

Deployed in 2005, Identified in 2010



STUXnet Worm

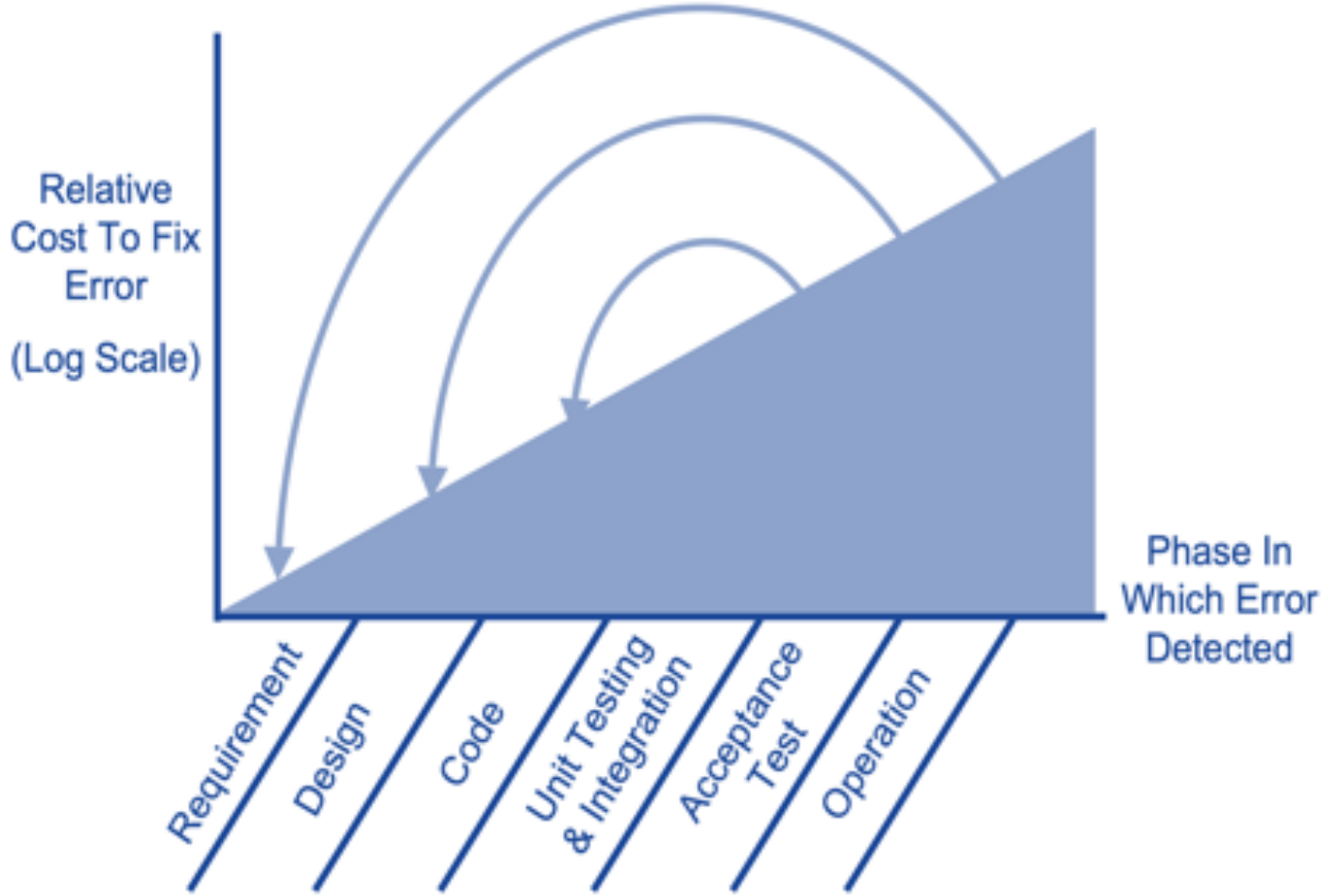
August 2003

Northeast Power Blackout

August 2003

Davis-Besse Nuclear Power Plant

Cost of Delayed Error/Bug Detection



1.
Incomplete requirements
Bad requirements gathering and documentation.

2.
Poor communication
Poor communication between programmers, stake holders, users and management. User involvement missing.

3.
Lack of resources
Insufficient and poorly skilled developers. Developers that are not team players.

4.
Unrealistic goals
Unrealistic development schedule, leading to time pressures and developer stress.

5.
Changing requirements
Requirements that change frequently, feature creeps.

6.
Lack of planning
Inefficient project planning and inconsistent project documentation.

7.
Sloppy development practices
Development practices from the Stone Age. For example, no version control or bug tracking during development. Testing is usually an afterthought.

8.
Poor reporting
Poor reporting of the project's status. Stake holders do not have any idea of the projects progress and milestones.

9.
Use of immature technology
Using new technology to impress clients. Technology that developers are unfamiliar or poorly skilled with.

10.
Commercial pressures
Pressures to launch the product for commercial reasons.

10 Reasons why software projects fail.

by Sameer Borate
www.codediesel.com

How the customer explained it

How the project leader understood it

How the engineer designed it

How the programmer wrote it

How the sales executive described it

How the project was documented

What operations installed

How the customer was billed

How the helpdesk supported it

What the customer really needed

Why Software Projects Fail?

- Modules that do not fit together.
- Software that are hard to maintain/extend.
- Poor quality
- Unacceptable performance.
- Technology change and team-member change over time in long period projects.

What is a Successful Software Project?

- Deliver the **required functionality**. (Functional Requirements)
- non-Functional Requirements:
  - **Efficient**: does not waste valuable resources and response time.
  - **Usable**
  - **Dependable**: *reliable*, *secure*, and *safe*.
  - **Maintainable**
  - Within *budget* and *time*.

In-class Activity II

Good Attributes of Software

Activity 1: *Good Attributes of Software*

1. A medical system that has a **failure rate** of 2% in a year.
2. A bank security system that has a 95% **reliability**.
3. A university registration system that requires 1 day of student **training** before students can use it.
4. An online banking system that serves a national bank and allows **1000 concurrent users** to access the system.
5. A complex train station control system that needs 3 months of **training** before administrators can operate it.
6. A national polling system that has a **reliability** of 96%.

Activity 1: *Good Attributes of Software*

7. A heart-monitoring unit that is used in a hospital's intensive care has a heart attack detection function that is required to have a **failure rate** of less than one per million cases.
8. One requirement of a supermarket billing software system that it *should not fail, on average, more than 10 minutes per month during the* ***supermarket's working hours***. In addition, the probability that the off-time (the time needed for repair and recovery of all the supermarket services) be more than 30 minutes is required to be less than 0.5%.

Activity 2: *Good Attributes of Software*

- You are working as a software consultant in university X. You are responsible for providing recommendation on buying one of the following new software systems for managing university student registration:
  A. The first system costs only $100k but requires an additional cost of $1k for annual system support. It also requires 5 days training and is upgraded yearly to newer version.
  B. The second system costs only $40k but requires an additional cost of $3k for annual system support. It also requires 3 days user training and is upgraded every 2 years to newer version.
- In your opinion, which of the two systems has:
  - higher maintainability
  - higher dependability
  - higher efficiency
  - higher acceptability
  - higher usability

In-class Activity III

Comprehensive Software Requirements

Goal: *Comprehensive Software Requirements*

- Read the following sample real-world software development cases and try to answer the following questions:
  - Identify the functional and non-functional requirement for each case.
  - Summarize the real causes of the mentioned issues for each case.
  - List the specific requirements that are missing and/or were not specified.

Case 1: *Sales Information System*

"Our new sales information system seems okay, the invoices are correct, the inventory records are correct, the discounts granted to our clients exactly follow our very complicated discount policy, **but** our new sales information system frequently fails, usually at least twice a day, each time for twenty minutes or more. Yesterday it took an hour and half before we could get back to work . . . . Imagine how embarrassing it is to store managers . . . . Softbest, the software house that developed our computerized sales system, claims no responsibility . . . ."

Case 2: *Education Management System*

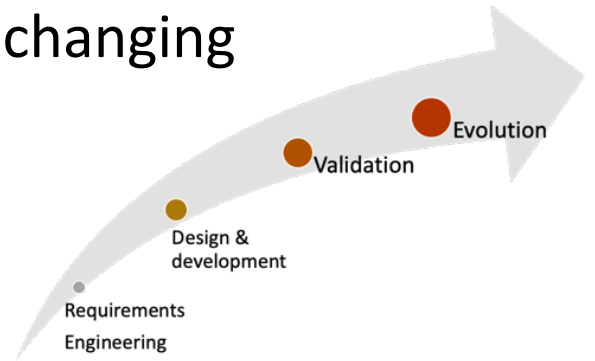
"Believe it or not, our software package 'Blackboard' for schoolteachers, launched just three months ago, is already installed in 187 schools. The development team just returned from a week in Hawaii, their vacation bonus. **But** we have been suddenly receiving daily complaints from the 'Blackboard' maintenance team. They claim that the lack of failure-detection features in the software, in addition to the poor programmer's manual, have caused them to invest more than the time estimated to deal with bugs or adding minor software changes that were agreed as part of purchasing contracts with clients."

Case 3: *Loan Contract Software*

"The new version of our loan contract software is really accurate. We have already processed 1200 customer requests, and checked each of the output contracts. There were no errors. **But** we did face a severe unexpected problem – training a new staff member to use this software takes about two weeks. This is a real problem in customers' departments suffering from high employee turnover . . . . The project team says that as they were not required to deal with training issues in time, an additional two to three months of work will be required to solve the problem."

Software Development Process

- A **systematic approach** for software development. It is a **sequence of activities** that leads to the *production of software*.
- Four fundamental activities are common to all software processes:
  - **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.



General Issues that Affect Software

- **Heterogeneity** – Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
- **Business and social change** – Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.
- **Security and trust** – As software is intertwined with all aspects of our lives, it is essential that we can trust that software.
- **Scale** – Software has to be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-scale, cloud-based systems that serve a global community.
- **Legacy systems** - old, valuable systems must be maintained and updated.
- **Dependability and Delivery** - having trustworthy software with faster delivery of software (time-to-market)

To address these challenges, we will need new tools and techniques as well as innovative ways of combining and using existing software engineering methods.

Software Engineering Diversity

- Software engineering is a systematic approach to the production of software that takes into account practical cost, schedule, and dependability issues, as well as the needs of software customers and producers.
- 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*.

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.
- **Batch processing systems**

  These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.

Application Types – Cont'd

- **Entertainment systems**

  These are systems that are primarily for personal use and which are intended to entertain the user.

- **Systems for modelling and simulation**
  - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
- **Data collection systems**
  - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
- **Systems of systems**
  - These are systems that are composed of a number of other software systems.

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.

- These fundamental notions of **process**, **dependability**, **requirements**, **management**, and **reuse** are important themes. Different methods reflect them in different ways, but they underlie all professional software development.

Internet Software Engineering

- The Web is now a platform for running application and organizations are increasingly developing web-based systems rather than local systems.
- Before the web, business applications were mostly **monolithic** running on single computers. Communications were local, within an organization.
- Now, software is **highly distributed**. Business applications are not programmed from scratch but involve reuse of components and programs.
- Web services (discussed in later in the course) allow application functionality to be accessed over the web.
- Cloud computing is an *approach to the provision of computer services where applications run remotely on the 'cloud'.*
  - Users do not buy software but pay according to use.

Web-based Software Engineering

- Web-based systems are complex **distributed** systems but the fundamental principles of software engineering discussed previously are as applicable to them as they are to any other types of system.
- The fundamental ideas of software engineering apply to web-based software in the same way that they apply to other types of software system.

Web Software Engineering

- **Software reuse**

  Software reuse is the dominant approach for constructing web-based systems. When building these systems, you think about how you can assemble them from pre-existing software components and systems.

- **Incremental and agile development**

  Web-based systems should be developed and delivered incrementally. It is now generally recognized that it is impractical to specify all the requirements for such systems in advance.

- **Service-oriented systems**

  Software may be implemented using service-oriented software engineering, where the software components are stand-alone web services.

- **Rich interfaces**

  Interface development technologies such as AJAX and HTML5 have emerged that support the creation of rich interfaces within a web browser.

Case Studies

Case Studies

- **A personal insulin pump**

  An embedded system in an insulin pump used by diabetics to maintain blood glucose control.
- **Mentcare: a mental health case patient management system**

  A system used to maintain records of people receiving care for mental health problems.
- **A wilderness weather station**

  A data collection system that collects data about weather conditions in remote areas.
- **iLearn: a digital learning environment**

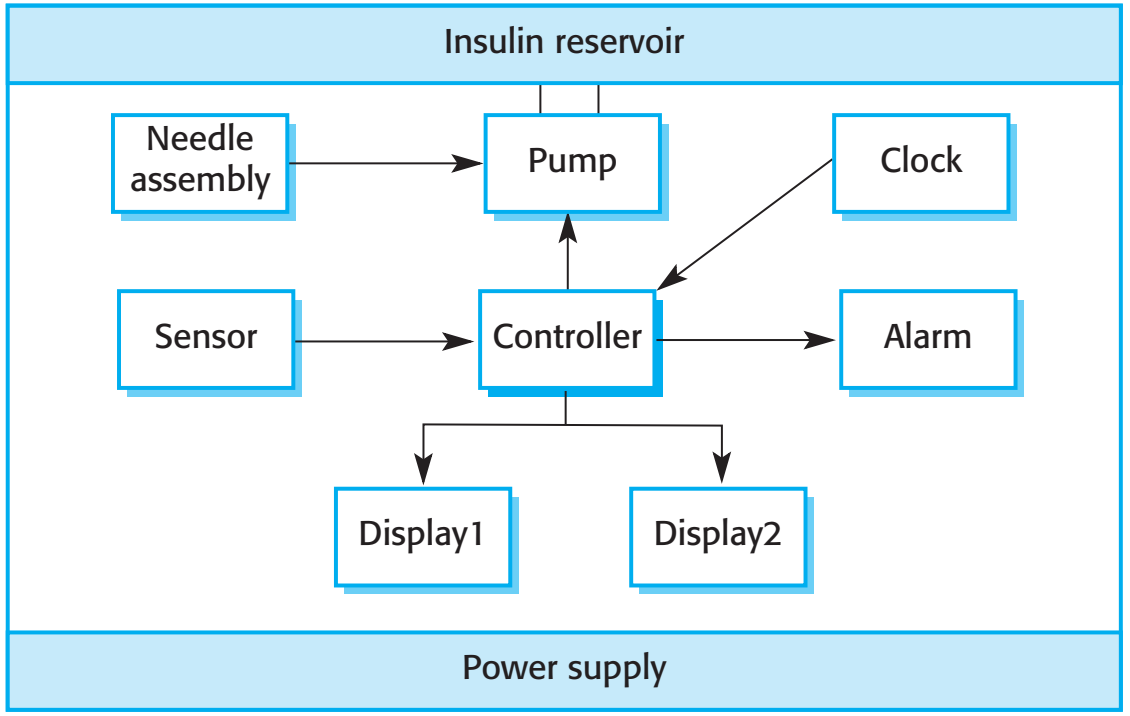
  A system to support learning in schools

Insulin Pump Control System

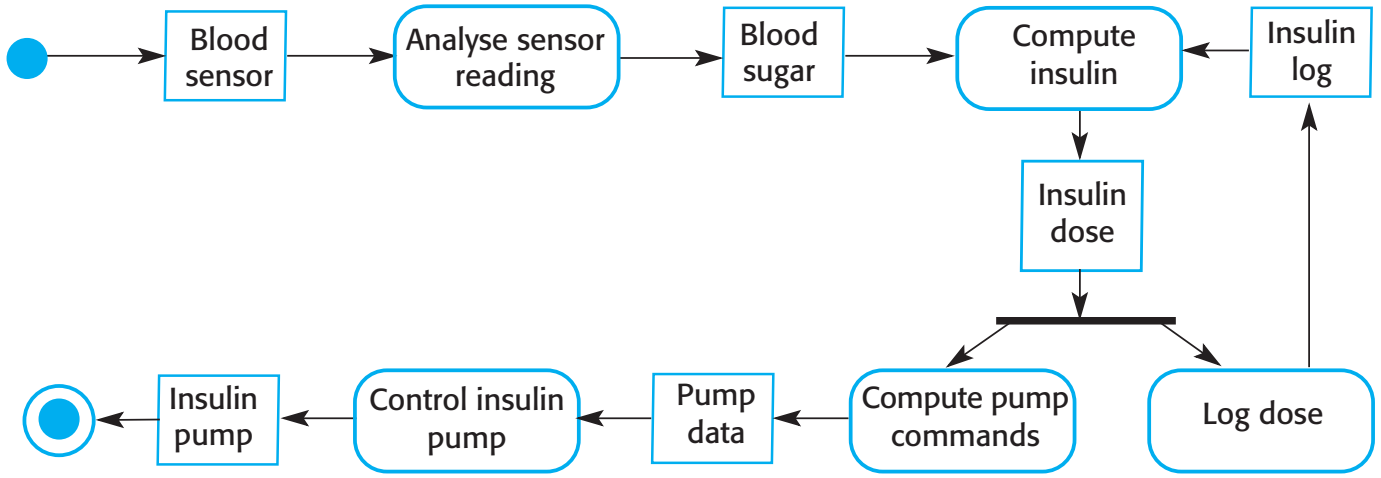
- Collects data from a blood sugar sensor and calculates the amount of insulin required to be injected.
- Calculation based on the rate of change of blood sugar levels.
- Sends signals to a micro-pump to deliver the correct dose of insulin.
- Safety-critical system as low blood sugars can lead to brain malfunctioning, coma and death; high-blood sugar levels have long-term consequences such as eye and kidney damage.

Insulin Pump Hardware Architecture

- A software-controlled insulin delivery system uses a microsensor embedded in patient to measure some blood parameter that is proportional to the sugar level.
- This is then sent to the pump controller to compute the sugar level and the amount of insulin that is needed.
- It then sends signals to a miniaturized pump to deliver the insulin via a permanently attached needle.



UML Activity Model of the Insulin Pump



illustrates how the software transforms an input blood sugar level to a sequence of commands that drive the insulin pump

Essential High-Level Requirements

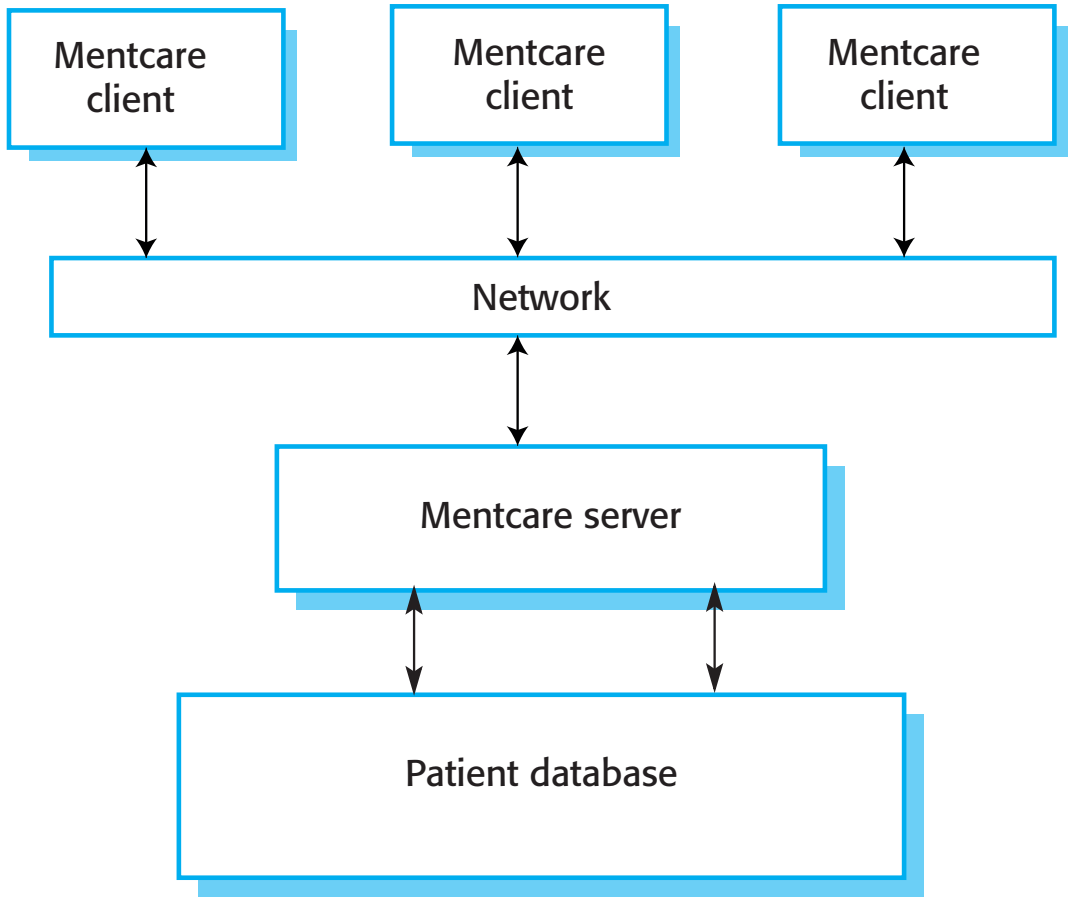
- The system shall be available to deliver insulin when required.
- The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.
- The system must therefore be designed and implemented to ensure that the system always meets these requirements.

Mentcare: *A patient information system for mental health care*

- A patient information system to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems.
- To make it easier for patients to attend, these clinics are not just run in hospitals. They may also be held in local medical practices or community centres.

Mentcare

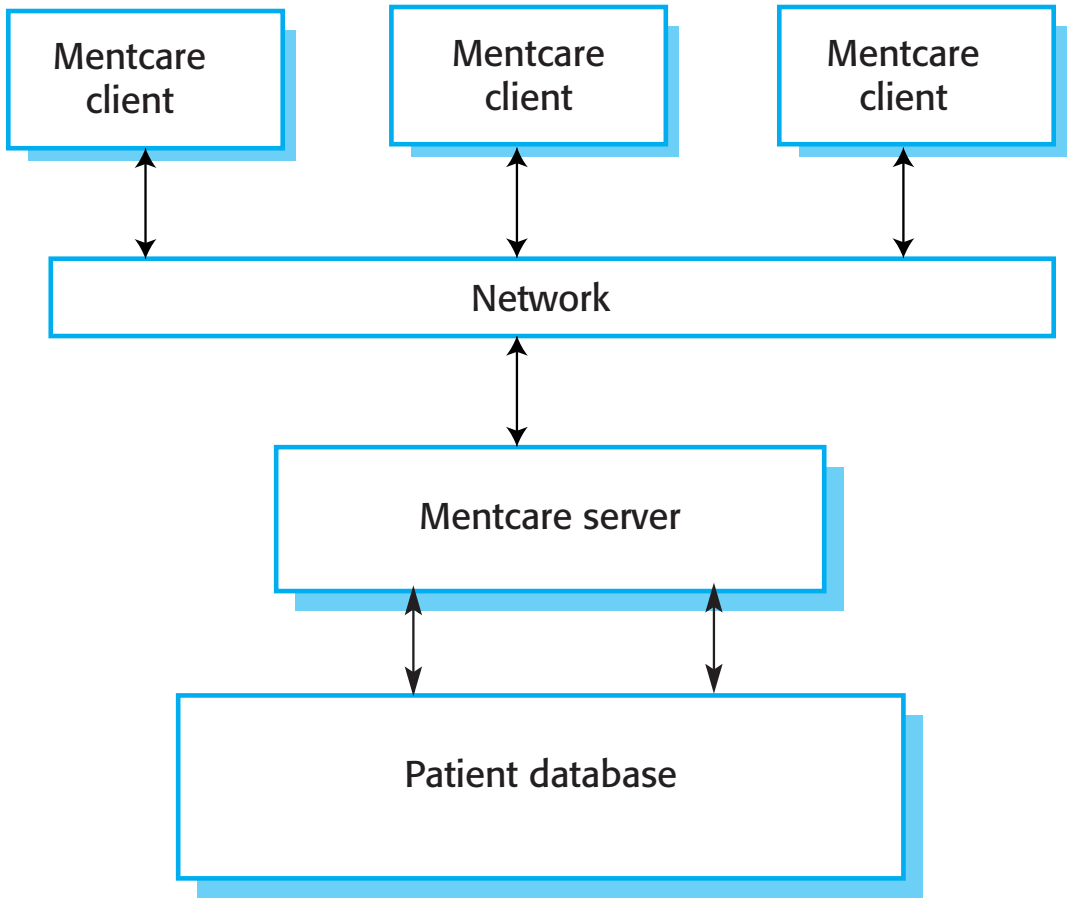
- Mentcare is an information system that is intended for use in clinics.
- It makes use of a **centralized database** of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.



Mentcare Goals

- To generate management information that allows health service managers to assess performance against local and government targets.
- To provide medical staff with timely information to support the treatment of patients.

Organization of the Mentcare System



Key Features of the Mentcare System

- **Individual care management**

  Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.

- **Patient monitoring**

  The system monitors the records of patients that are involved in treatment and issues warnings if possible problems are detected.

- **Administrative reporting**

  The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

Mentcare System Concerns

- **Privacy**
  - It is essential that patient information is confidential and is never disclosed to anyone apart from authorised medical staff and the patient themselves.
- **Safety**
  - Some mental illnesses cause patients to become suicidal or a danger to other people. Wherever possible, the system should warn medical staff about potentially suicidal or dangerous patients.
  - The system must be available when needed otherwise safety may be compromised and it may be impossible to prescribe the correct medication to patients.

Wilderness Weather Station

- The government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.
- Weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed and wind direction.
  - The weather station includes a number of instruments that measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure and the rainfall over a 24-hour period.
  - Each of these instruments is controlled by a software system that takes parameter readings periodically and manages the data collected from the instruments.

Weather Information System

- **Weather Station System**

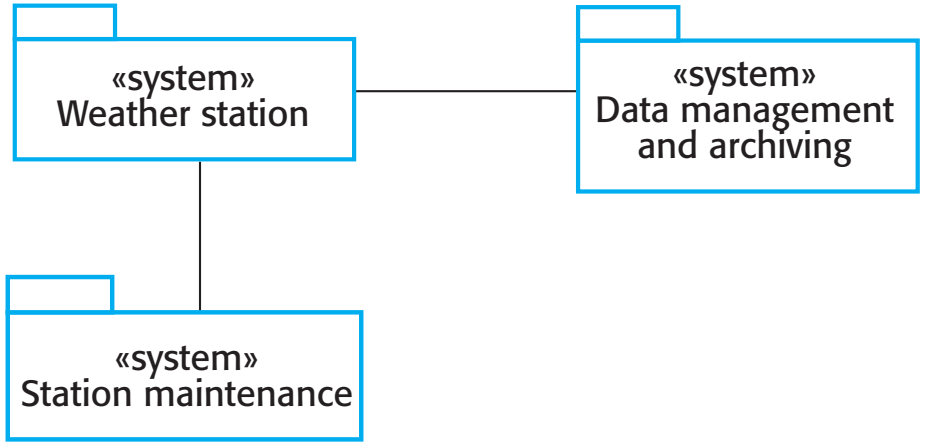
  This is responsible for collecting weather data, carrying out some initial data processing and transmitting it to the data management system.

- **Data Management and Archiving System**

  This system collects the data from all of the wilderness weather stations, carries out data processing and analysis and archives the data.

- **Station Maintenance System**

  This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems.



Additional Software Functionality

- Monitor the instruments, power and communication hardware and report faults to the management system.
- Manage the system power, ensuring that batteries are charged whenever the environmental conditions permit but also that generators are shut down in potentially damaging weather conditions, such as high wind.
- Support dynamic reconfiguration where parts of the software are replaced with new versions and where backup instruments are switched into the system in the event of system failure.

iLearn: *A digital Learning Environment*

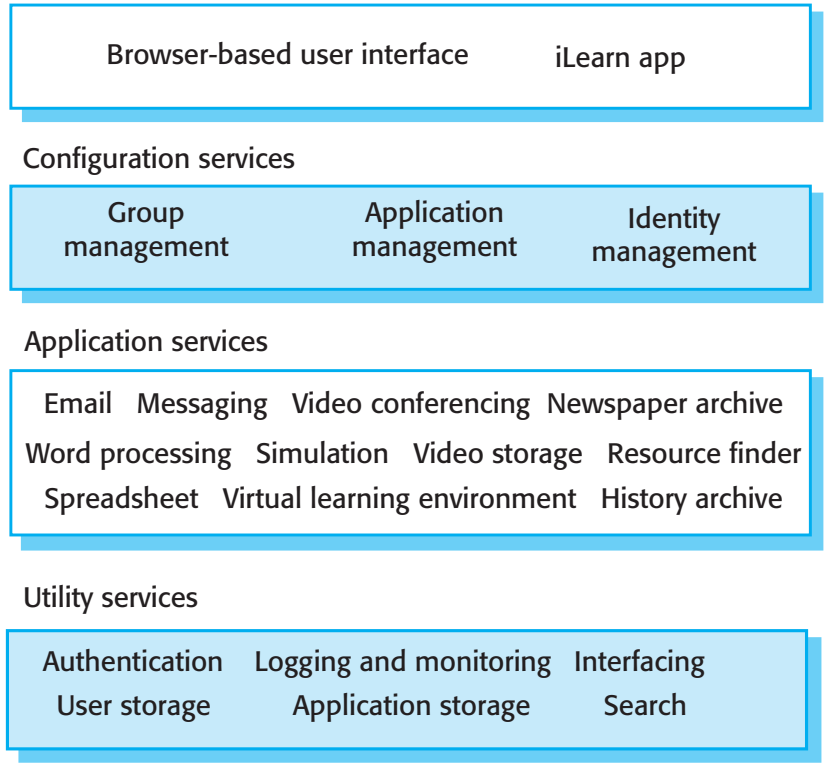
- A digital learning environment is a framework in which a set of general-purpose and specially designed tools for learning may be embedded plus a set of applications that are geared to the needs of the learners using the system.
- The tools included in each version of the environment are chosen by teachers and learners to suit their specific needs.
  - These can be general applications such as spread sheets, learning management applications such as a Virtual Learning Environment (VLE) to manage homework submission and assessment, games and simulations.

iLearn: *A Service-Oriented System*

- The system is a service-oriented system with all system components considered to be a replaceable service.
- This allows the system to be updated incrementally as new services become available.
- It also makes it possible to rapidly configure the system to create versions of the environment for different groups such as very young children who cannot read, senior students, etc.

iLearn Services

- ***Utility services*** that provide basic application-independent functionality and which may be used by other services in the system.
- ***Application services*** that provide specific applications such as email, conferencing, photo sharing and access to educational content such as scientific films or historical resources.
- ***Configuration services*** that are used to adapt the environment with a specific set of application services and do define how services are shared between students, teachers and their parents.



iLearn: *Service Integration*

- The environment has been designed so that services can be replaced as new services become available and to provide different versions of the system that are suited for the age of the users.
- This means that the system has to support two levels of service integration:
  - *Integrated services* are services which offer an API (application programming interface) and which can be accessed by other services through that API. Direct service-to-service communication is therefore possible.
  - *Independent services* are services which are simply accessed through a browser interface and which operate independently of other services. Information can only be shared with other services through explicit user actions such as copy and paste; re-authentication may be required for each independent service.

Generic Technical Terms

- **Notation** is a graphical or textual set of rules for representing a model.
  - Examples: Unified Modeling Language (UML) and Architecture Analysis and Design Language (AADL).
- **Method** is a repeatable technique that specifies the steps involved in solving a specific problem.
  - Examples: different sorting algorithms and heuristics.
- **Methodology**: a collection of methods for solving a specific set of problems.
  - Examples:

Computer-Aided Software Engineering

- CASE are software systems which are intended to provide *automated support for software process activities*, such as **requirements analysis**, **system modelling**, **debugging** and **testing**.
- Upper-CASE are the Tools to support the early process activities of requirements and design
- Lower-CASE are the tools to support later activities such as programming, debugging and testing