Lancaster University

SCC.211 Software Design

(Software Requirements)

Andrew Sogokoń: a.sogokon@lancaster.ac.uk

Ran Wei: r.wei5@lancaster.ac.uk

Gerald Kotonya (convenor) – g.kotonya@lancaster.ac.uk

School of Computing and Communications

InfoLab21

Lancaster University

Today's focus

- Some House keeping …

- About SCC211…
- Software requirements in the context software development
- Defining software requirements
- Guidelines for writing requirements
- Functional and non-functional requirements
- Requirements engineering

Plagiarism

Lancaster University

- Existing university policies on academic malpractice and plagiarism apply offline and online.
- Direct sharing of code, sharing solutions and/or partial solutions with other students, either privately or in an open chat, is **not acceptable**
- **Plagiarism** includes:
  - Submitting (e.g.) code that someone else wrote
  - Paying for someone else to do it for you
  - Working on a piece of in work together as a group, and submitting it as individual work
  - Sharing of code that you then possibly adapt

Some ways to avoid being a victim of plagiarism

Lancaster University

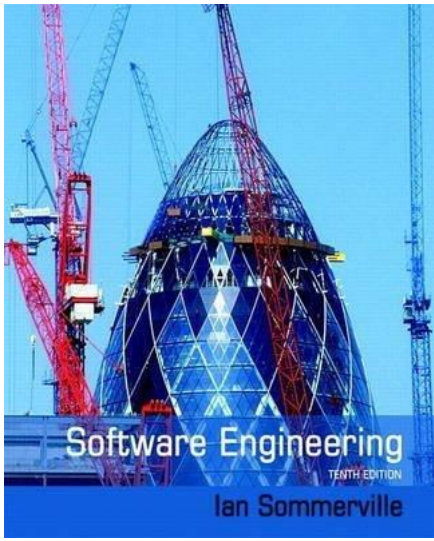
- Give attribution to ideas, text, code, algorithms, etc., that you have sourced elsewhere by crediting/citing the source.
- If you use code libraries that are not part of the standard language library, credit/cite the source
- Make your GitHub repo private
- **Examples of attribution …**
  - A study in 2020 found that *Programming langX* had overtaken *Programming langY* as the most popular programming language [**1**].
  - My implementation used the statistical library developed by J.R. Hart [**2**]

References

1. Joe Bloggs, "A Review of Programming Languages", *Journal of Software Dev*, publisher name, **pp**. 2-6, 2020.
2. J.R. Hart, http://www.libsources.com, accessed 10th October 2022.

What we expect from you

- Integrity (no plagiarism, no faking results) and effort (active learning):
  - come to the workshops
  - get the suggested textbook (Sommerville 10th ed)
  - take notes (evidence handwritten are better)
  - read about subject/try things for yourself
  - ask us questions in the workshop sessions
  - plan your time and coursework carefully



What you can expect from us

Lancaster University

- We'll do our best
  - to make all our notes available on Moodle
  - to give you references to follow up
  - to arrange extra support if you've already tried the normal routes (book, web, forum, TAs) – *Talk to the Teaching Office.*
  - to offer feedback on formative coursework promptly
  - to respond to email (ideally as a last resort! - *note: we get more email than we can handle* and have a lot of teaching/research/admin commitments, so are often *not* in our offices!)

About SCC.211 …..

Lancaster University

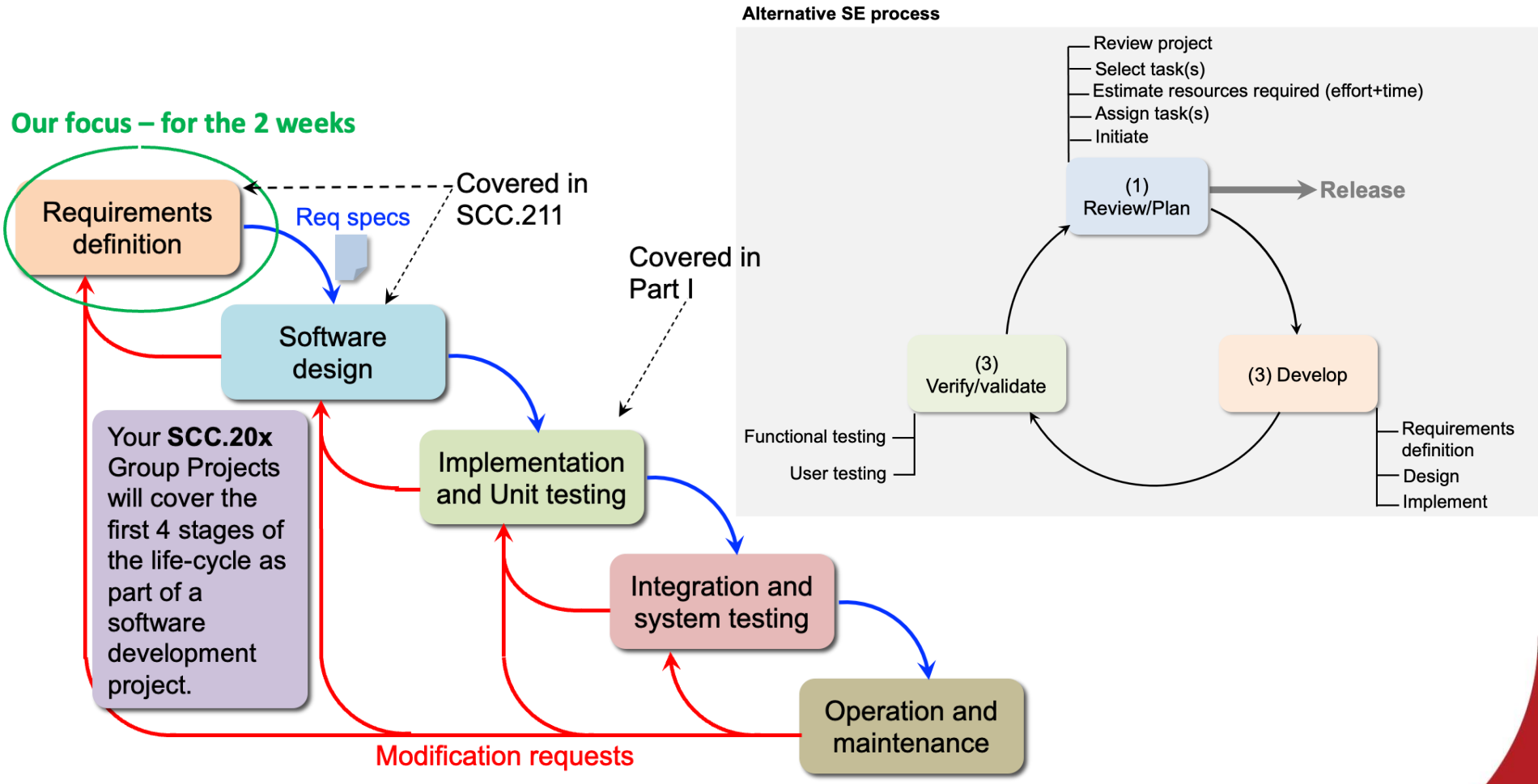
- **Objectives**
  - To introduce you to software requirements, use case modelling, & software architecture (**Andrew**)
  - To introduce you to object-oriented software design through (Class diagrams), SOLID design principles, and dynamic design (State diagrams) (**Ran**)
  - To introduce you to software design patterns (**Gerald**)
- **Structure**
  - **Weeks 1-3**: Software Requirements, Use case modelling, Software architecture (**Andrew**)
  - **Week 4-6**: Object-oriented design, SOLID principles, State diagrams (**Ran**)
  - **Week 7-9:** Design patterns (**Gerald**)
  - **Week 8:** **SCC211 Test** **(All)**
  - **Week 10**: Exam revision lecture (**All**)
    - Will look at the structure and content of the exam, and review some past exam questions

Lectures, Workshops & Assessment

Lancaster University

- Lectures & Workshops
  - Wednesday lectures: **Wed** (9-10:00) and **Wed** (17-18:00) will be used to introduce new topics
  - The lectures will be followed by problem workshops.
  - Friday lectures: **Fri** (14:00-15:00) and **Fri** (15:00-16:00) will be used to present and to discuss solutions to the workshop problems
- Assessment/coursework (in-lab test)
  - There will be **one** coursework for this module
  - The coursework will be in the form of an **in-lab** multiple choice question test
  - The in-lab test will be done in **Week 8**, in your workshop session
  - The coursework mark will be worth **30%** of the final module mark. The **summer** exam will contribute **70%** to final module mark
- In **Week 5**, we will provide you with a coursework problem.
  - The problem will cover all the material covered up to that point.
  - The in-lab test will be based on the coursework

Software development life-cycle -**recap**



Software engineering stack

Lancaster University

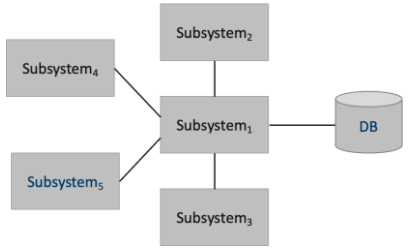
Management
(Software evolution)

Software Requirements Definition
(Software Modeling)
(elicitation, analysis, documentation)

Software Design
(Software Modeling)
(architecture, interface, component, algorithm design)

Software Implementation
(Software Construction)
(Code production)

Verification & validation
(Software evaluation)



Simple software architecture

What is a software requirement?

- A software requirement is a description of **what** the system under development should do and **any constraints** under which it must do it
  - Consider the example of an Automated Teller Machine(ATM) or (Cashpoint)
- Other definitions ….
  - The **IEEE Standard Glossary** of Software Engineering Terminology defines a requirement as:
    1. A condition or capability needed by a user to solve a problem or achieve an objective using a software system.
    2. A condition or capability that must be met or possessed by a software system or system component to satisfy a contract, standard, specification, or other formally imposed document.
    3. A documented representation of a condition or capability as in (1) or (2).

Why are software requirements important?

- They define the software's purpose (scope the project)
- They serve as a contract between the development team and the customer
- They are fundamental to the verification and validation of the system.
  - Enable engineers to create test cases that cover all aspects the software they are developing
- They guide the development process
  - Act as a roadmap for the development team, ensuring the software meets it's intended purpose.
  - Fixing requirements related errors at the implementation stage costs **10x** more than at requirements

Guidelines for writing requirements

- Use language in a consistent way.
  - Use **shall** for mandatory requirements, **should** for desirable/useful requirements.
  - Consider an App for buying train tickets. The App might have the following three requirements:
    - The App **SHALL** enable the user to buy trains tickets (**mandatory**)
    - The App **SHALL** support accessibility in accordance with the 2018 accessibility regulation (**mandatory**)
    - The App **SHOULD** allow users to share their experience of using it
- Use text highlighting to identify key parts of the requirement.
- Don't make assumptions about the reader's knowledge.
- Involve as many **stakeholders** as possible
- Include an explanation (rationale) of why a requirement is necessary.

Requirements and the customer

Lancaster University

- Requirements can be written at different levels of abstraction or detail to suit different stakeholders.
- Customers do not need the same level of detail as the technical or development team.
  - So, it is important that requirements are written at level that the customer understands
  - User requirements, intended for the customer, are best written using natural language, tables and diagrams

Different levels of abstraction in software requirements

Lancaster University

- **User requirement**

  1. The patient management system **shall generate** monthly management reports showing the cost of drugs prescribed by each clinic during that month

- **System requirement**

  1.1. On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics **shall be generated**.

  1.2. The system **shall automatically generate** the report for printing after **17.30** on the last working day of the month.

Separate Design considerations Requirements

- A software requirement focuses on **what** the system will do, not **how** it will do it
- Consider the following requirements:
  1. The train **shall** transport 5,000 passengers per hour from Lancaster to Manchester ✓ What
  2. The train **shall** consist of 3 carriages pulled by an electric locomotive ✗ How
  3. The system development costs **shall** be less than £5million
  4. The system impact on the environment **shall** be B+ rated or higher

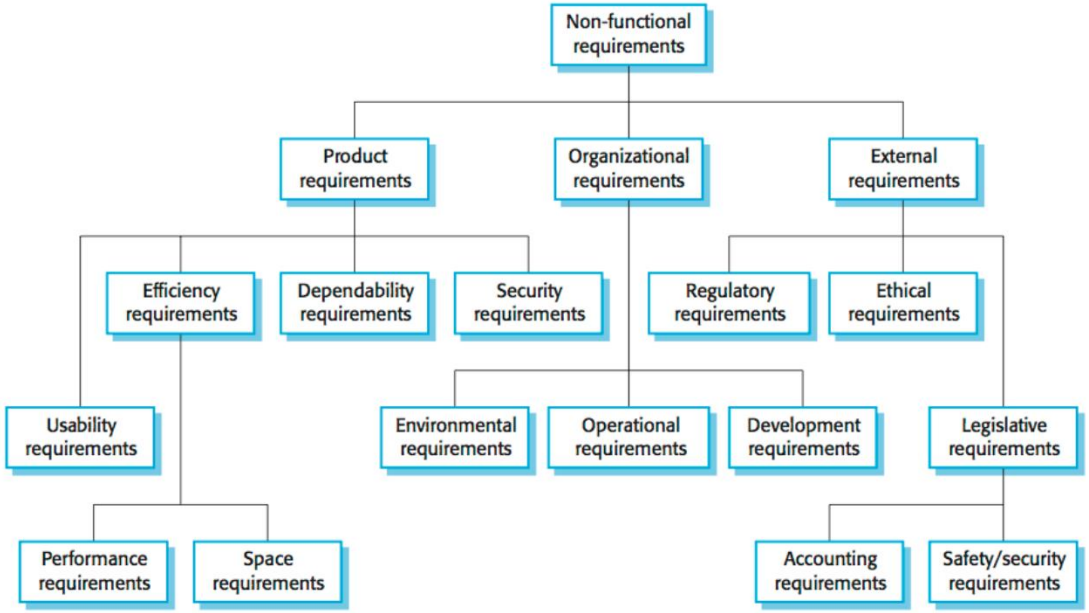
- Design considerations might be:
  - Type of locomotive (electric, diesel, etc.); type of track; number & type of carriages; design of carriages (e.g., type of seats); etc.

Are the following good or bad requirements?

1. “The telephone system shall be able to inform callers of the anticipated wait time and their number on the wait queue”
2. “System shall support **client** enquiries using three access points: in person, paper-based mail and voice communication
3. “The telephone system shall be **able** to support an **0800** number system to allow free calls”
4. “If an individual double-clicks on an event in a member’s journal, the system shall display the images associated with an event”
5. “The telephone system shall be able to handle 97,000 calls per year and must allow for a growth rate of 15 percent annually”

Types of software requirements

- There are two main types of software requirements
  - Functional requirements (FR) are requirements for services that a system should deliver.
    - For example, *The ATM system **shall** enable users to withdraw cash*
  - Non-functional requirements (NFR) specify the quality with which a system delivers its services. Typical examples include:



Three classes of non-functional requirements:

- NFRs are **SHALL** statements that precisely specify these constraints.

Metrics for specifying Non-functional requirements

Lancaster University

Property	Measure
Speed (Performance)	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes No of ROM (Read Only Memory) chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure (MTTF) Probability of unavailability Rate of failure occurrence
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Number of target systems

Functional or Non-functional requirement?

1. The train shall be able to transport passengers from Lancaster to Manchester.
2. The capacity shall be at least 5000 passengers per hour.
3. The development costs shall be less than £5 million.
4. The operating costs shall be less than £150,000 per day.
5. Late arrivals shall occur less than 1% of the time.
6. The impact on the environment should be B+ rated or higher.

** It is important to ensure that requirements are **testable**

- "The system shall be user-friendly" ….. Is difficult to verify

Requirements document

Lancaster University

- Usually a text document
  - As described in the **IEEE Standard 830**
- Includes:
  - A purpose
  - Scope of the product
  - A set of **SHALL** statements prescribing what the system will do
    - both *functional* and *non-functional*

Example **shall** statements: Automated Teller Machine (ATM)

1. The ATM system shall enable a user to withdraw cash (F)
2. The ATM system shall enable users to check their account balance (F)
3. The ATM services shall be available for at least 23.5 hours daily (NF)
4. The ATM system shall be completed and installed in 6 months from the start of the project (NF)

Example of IEEE Standard 830 requirement specification template

Lancaster University

A.1 Template of SRS Section 3 organized by User classes:

3. Specific requirements

3.1 External interface requirements

- **3.1.1** User interfaces
- **3.1.2** Hardware interfaces
- **3.1.3** Software interfaces
- **3.1.4** Communications interfaces

3.2 Functional requirements

- **3.2.1** User class 1
  - **3.2.1.1** Functional requirement 1.1
  - :
  - **3.2.1.*n*** Functional requirement 1.*n*
- **3.2.2** User class 2
- :
- **3.2.*m*** User class *m*
  - **3.2.*m*.1** Functional requirement *m*.1
  - :
  - **3.2.*m*.*n*** Functional requirement *m*.*n*

3.3 Performance requirements

3.4 Design constraints

3.5 Software system attributes

3.6 Other requirements

See page 21-26 of:

- https://ieeexplore.ieee.org/document/720574

Requirements engineering

Lancaster University

- A systematic process of ….
  - Understanding the problem
  - Specifying the solution
  - And once the solution has been **validated**
    - Managing the requirements

Understanding the problem

Lancaster University

- It is important to to understand the context of problem that the system is intended to address
  - What is the business goal?
  - Who are the proposed system users and other stakeholders?
  - What is the operational environment?
  - “Green field” or evolution of existing system?
  - What can we change, what must we keep?

Specifying the solution

Lancaster University

- Specify the proposed system by addressing the following questions:
  - What is needed to:
    - Achieve the business goal?
    - Support the users?
  - How do these requirements interact?
    - Some may conflict
    - Some will be higher priority than others (essential, useful, desirable)
  - Are the requirements you have formulated the right requirements?

Stakeholders

Lancaster University

- You need to identify the important stakeholders in a system to discover their requirements
- System stakeholders: Any person or organisation who is affected by the system in some way and so has a legitimate interest
- They include
  - End users of the system
  - Managers and others involved in the organisational processes influenced by the system
  - Engineers responsible for the development and maintenance
  - People responsible for systems that will interface with the system under development
  - External bodies such as regulators or certification authorities

Stakeholders – An Example

Lancaster University

Consider an information system that is intended for use at a GP practice. We can identify a number of system stakeholders ….

- Patients whose information is recorded in the system
- Doctors who are responsible for assessing and treating patients.
- Nurses who coordinate the consultations with doctors and administer some treatments.
- Medical receptionists who manage patients' appointments.
- IT staff who are responsible for installing and maintaining the system.
- Legislation on Data Protection
- NHS database
- GP Practice

Why requirements are managed

Lancaster University

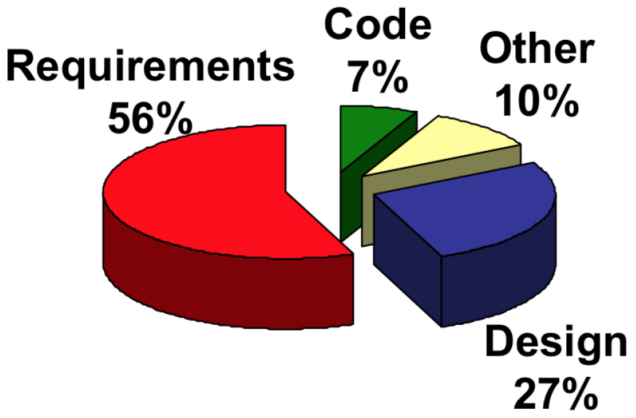
- Requirements beget requirements …
  - *Business goals -> users requirements –> software requirements – software components*
- We want to track them
- We might have to reappraise the requirements
- Requirements change

Motivation

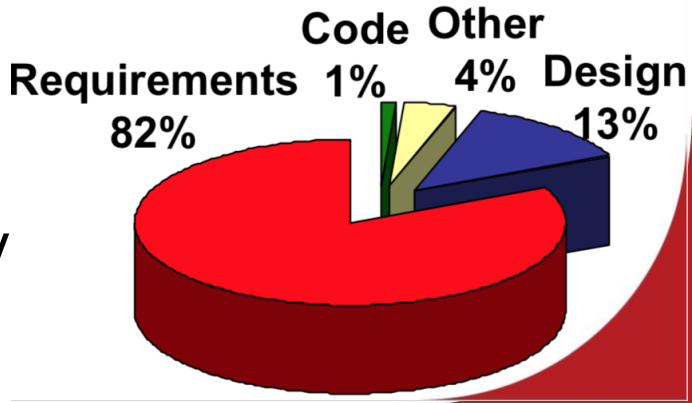
Lancaster University

- Requirements related errors are... the most numerous of software errors
- The most persistent
- The most expensive: account for **60%** of software error costs
  - **~5x** more if fixed during design
  - **~10x** more if fixed during
  - implementation
  - **~20x** more if fixed during integration testing
  - **~100-200x** more if fixed after delivery

Distribution of Defects



Distribution of effort to fix defects



Summary

Lancaster University

- **What is a requirement?**
- A description of what the system under development (SUD) should do and any constraints under which it must do it
- **What is requirements engineering?**
- The process involved in developing system requirements
- **Who are the system stakeholders?**
- Anyone who is affected in some way by the system. People or organisations who have an ‘interest’ or ‘stake’ in the system
- **What happens when requirements are wrong?**
- Systems are late, unreliable and don’t meet customer expectations

Suggested reading

Lancaster University

- Software Engineering (10th Ed.), Ian Sommerville
  - Chapter1 “Introduction” and
  - Chapter4 “RequirementsEngineering”
- Integrated Requirements Engineering: A Tutorial, by Ian Sommerville

http://www.cs.rug.nl/search/uploads/Teaching/RE2009Fall/paper/2005_Sommerville_IEEE%20Software_Integrated%20Requirements%20Engineering%20A%20Tutorial.pdf

Lancaster University

Thank you!