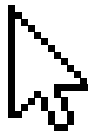
COE 454
Software Engineering II:

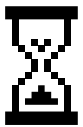
Introduction



Presented by Eliel Keelson

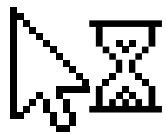
“For which of you, desiring to build a tower, does not first sit down and count the cost, whether he has enough to complete it?.”

—The Famous Jesus (Luke 14:28)



What You Need to Pass this Course

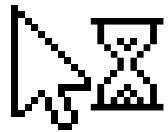
- Punctual Attendance
- Attentiveness
- Class/Online Participation
- Timely Submission of Assignments
- Group Work
- Honesty




Overview of Software Engineering

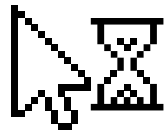
Introduction

- Software Engineering is a disciplined and systematic approach to the design, development, maintenance, and testing of software applications.
- It encompasses a wide range of principles, methodologies, and tools aimed at producing high-quality software that meets user requirements and is reliable, maintainable, and scalable.



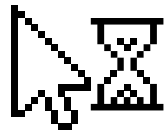
Relevance of SE

- In today's digital age, software is at the heart of almost every industry, driving innovation and efficiency.
- From healthcare systems and financial services to entertainment and communication, software applications are integral to the functioning of modern society.



Relevance of SE

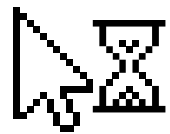
- Software Engineering cannot be overstated for several key reasons:
  1. **Critical Infrastructure:** Software systems underpin critical infrastructure in areas such as healthcare, finance, transportation, and energy. Ensuring these systems are reliable and secure is paramount.



Relevance of SE

2. **Economic Impact:** The software industry is a significant contributor to the global economy, creating jobs, driving innovation, and enhancing productivity across various sectors.

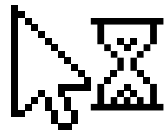
3. **Complexity Management:** Modern software systems are highly complex, requiring sophisticated engineering approaches to manage their development and maintenance effectively.



Relevance of SE

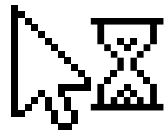
4. **Quality Assurance:** Ensuring software quality, including functionality, performance, security, and usability, is crucial for user satisfaction and business success.

5. **Innovation and Development:** Software Engineering fosters innovation, enabling the development of new technologies and solutions that address emerging needs and challenges.



Areas of Concern in SE

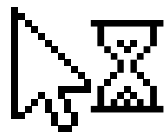
- To excel in Software Engineering, one must understand several fundamental concepts that form the foundation of the field:
  1. Software Development Life Cycle (SDLC)
  2. Software Development Methodologies
  3. Requirements Engineering
  4. Software Design and Architecture
  5. Testing and Quality Assurance
  6. Project Management
  7. Version Control and Configuration Management
  8. Software maintenance
  9. Ethics and professionalism




Software Development Lifecycle

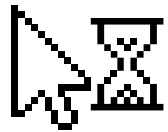
SDLC

- The Software Development Life Cycle (SDLC) is a structured process used by software developers to design, develop, and test high-quality software.



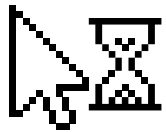
SDLC

- The aim of SDLC is to produce software that:
  - meets or exceeds customer expectations,
  - reaches completion within times and cost estimates,
  - and works effectively and efficiently in the current and planned information technology infrastructure.



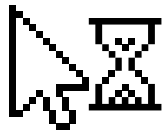
SDLC

- The traditional steps in the SDLC are:
  - Problem Definition
  - Requirements gathering and analysis
  - Planning and Designing
  - Development/Implementation
  - Testing
  - Deployment
  - Maintenance and Support



SDLC Assignment

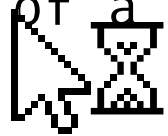
- Relate any four traditional software development strategies or methodologies to SDLC
- Using a well-drawn infographic, highlight the major pros and cons of these methodologies as a way of differentiating them.
- When are they often suitable?




Functional & Nonfunctional Requirements

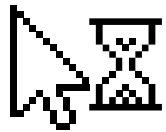
Introduction

- Up to this point, in your previous semester, we have established the essence of requirements and methods of eliciting these requirements from our problem domain.
- In our study, we focused on two main types of requirements
  - functional and
  - non-functional requirements
- These are two distinct categories of requirements that capture different aspects of a software system.



Functional Requirements

- Functional requirements describe the specific functionalities, features, and behaviours that the software system must exhibit to fulfil the needs and expectations of its users.
- These requirements typically define the interactions between the software and its users or other system components, specifying how the system should respond to various inputs or stimuli.

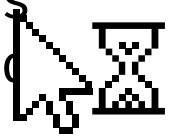


Non-Functional Requirements

- Non-functional requirements, also known as quality attributes or system qualities, focus on the characteristics of the software system that are not directly related to its specific functionalities.
- They define the desired qualities, constraints, and criteria that the system should exhibit in terms of performance, security, usability, reliability, maintainability, scalability, and other factors.

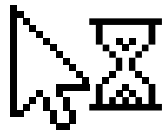
Why they are Important

- **Meeting User Expectations**: Functional requirements ensure that the software system delivers the desired functionalities, meeting the needs and expectations of the users or stakeholders.
- **System Reliability and Performance**: Non-functional requirements contribute to the reliability, performance, and overall quality of the software system. They address aspects such as speed, responsiveness, security, and robustness.



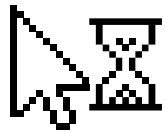
Why they are Important

- **System Design and Development**: Both types of requirements guide the design and development process. Functional requirements serve as a blueprint for system functionality, while non-functional requirements inform design decisions and help prioritise system qualities.



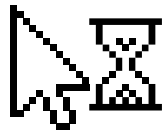
Why they are Important

- **Testing and Validation:** Functional requirements provide the basis for testing individual functionalities, while non-functional requirements guide testing efforts to validate system qualities.



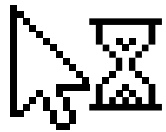
Why they are Important

- **System Evaluation and Acceptance:** Both types of requirements serve as criteria for evaluating the software system during the acceptance phase. Functional requirements ensure that the desired functionalities are met, while non-functional requirements assess system qualities and suitability for deployment.



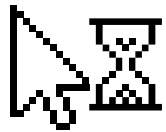
How to Elicit Functional Requirements

- Eliciting functional requirements involves gathering and documenting the specific functionalities and features that the software system needs to provide.
- Here are some commonly used techniques for eliciting functional requirements:



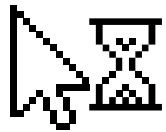
How to Elicit Functional Requirements

- **Stakeholder Interviews**: Conduct interviews with key stakeholders, such as clients, users, subject matter experts, and business analysts. Ask open-ended questions to understand their needs, expectations, and desired system functionalities.



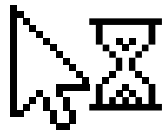
How to Elicit Functional Requirements

- **Workshops and Group Discussions:** Organize workshops or group discussions involving various stakeholders to brainstorm and gather requirements collaboratively. Facilitate discussions to uncover functional requirements, identify user scenarios, and clarify ambiguities.



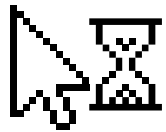
How to Elicit Functional Requirements

- **Document Analysis:** Analyze existing documents, such as business process documentation, user manuals, or system specifications, to identify potential functional requirements. Extract relevant information about desired functionalities, system interfaces, and user interactions.



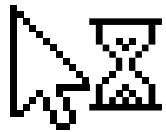
How to Elicit Functional Requirements

- **Prototyping and Mock-ups:** Create prototypes or interactive mock-ups to represent the proposed system visually. Use these prototypes to elicit feedback from stakeholders, which can lead to the identification of additional or refined functional requirements.



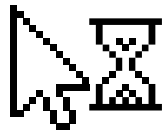
How to Elicit Functional Requirements

- **Use-Case Modeling:** Utilise use-case modelling techniques to capture system interactions and define user scenarios. Identify actors, system boundaries, and main use cases, and document the required functionality and system behaviour for each use case.



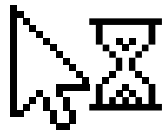
How to Elicit Functional Requirements

- **Observation and Job Shadowing:** Observe end-users or domain experts as they perform their tasks in their real-world environment. This allows you to gain insights into their activities, challenges, and requirements for the software system.



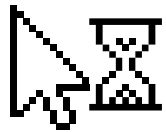
How to Elicit Functional Requirements

- **Observation and Job Shadowing:** Observe end-users or domain experts as they perform their tasks in their real-world environment. This allows you to gain insights into their activities, challenges, and requirements for the software system.



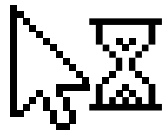
How to Elicit Functional Requirements

- **User Stories:** Employ user stories, typically used in Agile methodologies, to capture functional requirements from a user's perspective. These short narratives describe the desired system behaviour, focusing on who the user is, what they want to accomplish, and why it is important to them.



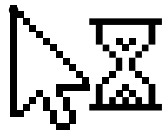
How to Elicit Non-Functional Requirements

- Eliciting non-functional requirements involves capturing the qualities, constraints, and criteria that define the desired attributes of the software system.
- Here are some effective techniques for eliciting non-functional requirements:



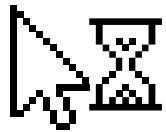
How to Elicit Non-Functional Requirements

- **Interviews and Questionnaires:** Conduct interviews or distribute questionnaires to stakeholders to gather their perspectives on non-functional requirements. Explore their expectations regarding system performance, security, usability, reliability, maintainability, and other relevant aspects.



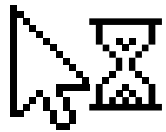
How to Elicit Non-Functional Requirements

- **Brainstorming Sessions:** Organise brainstorming sessions with stakeholders, including system users, business owners, technical experts, and subject matter experts. Encourage participants to share their insights and thoughts on non-functional requirements.



How to Elicit Non-Functional Requirements

- **Surveys:** Distribute surveys to gather feedback from a wide range of stakeholders. Use the surveys to collect their opinions on system qualities, such as performance, availability, and user experience.

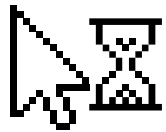


How to Elicit Non-Functional Requirements

- **Benchmarking and Comparative Analysis:** Conduct benchmarking activities to compare the system against similar existing solutions or industry standards. Identify the desired benchmarks and use the findings to elicit non-functional requirements related to performance, scalability, security, or other relevant areas.

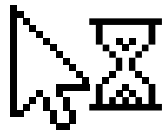
How to Elicit Non-Functional Requirements

- **Expert Consultation:** Seek advice and input from domain experts or specialists who have experience in similar systems or specific non-functional areas. Their expertise can help identify relevant non-functional requirements and guide discussions on system qualities.



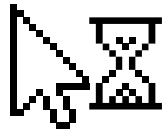
How to Elicit Non-Functional Requirements

- **Regulatory and Industry Standards Analysis:** Review applicable regulatory requirements and industry standards that pertain to the software system. Identify the non-functional requirements imposed by these regulations or standards and incorporate them into the requirements elicitation process.



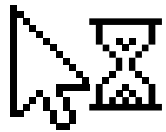
How to Elicit Non-Functional Requirements

- **Use Case Scenarios:** Utilize use case scenarios to explore non-functional requirements. Consider various scenarios and assess how the system should perform, considering aspects such as response time, availability, and security.



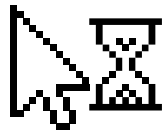
How to Elicit Non-Functional Requirements

- **Prototyping and Proof of Concept:** Develop prototypes or proof-of-concept solutions to validate and assess non-functional requirements. Use these prototypes to evaluate system qualities and gather feedback from stakeholders.



How to Elicit Non-Functional Requirements

- **Contextual Inquiry and User Observation:** Observe users in their work environment to understand how non-functional requirements, such as usability or efficiency, influence their tasks. Observe their behaviours, challenges, and pain points to identify relevant non-functional requirements.

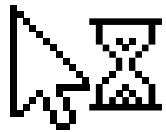


Let's consider a typical software engineering task of developing an online banking system. Here are some main functional and non-functional requirements worth considering:

Functional Requirements worth considering

1. **User Registration and Authentication**:

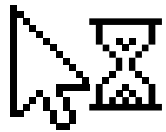
   - Users should be able to create new accounts, provide necessary information, and authenticate themselves securely.
   - The system should verify user credentials, such as usernames and passwords, to ensure secure access to the accounts.



Functional Requirements worth considering

2. **Account Management:**

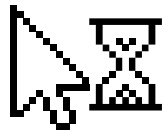
   - Users should be able to view their account balances, transaction history, and details of their accounts (e.g., savings, checking).
   - They should be able to perform actions such as making deposits, withdrawals, and fund transfers between accounts.



Functional Requirements worth considering

3. **Bill Payment:**

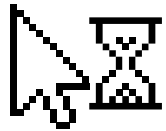
   - The system should provide a secure and convenient way for users to pay their bills online.
   - Users should be able to add billers, schedule payments, and receive confirmations of successful transactions.



Functional Requirements worth considering

4. **Fund Transfers:**

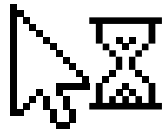
   - Users should be able to transfer funds to other accounts within the same bank or to external accounts in a secure and reliable manner.
   - The system should validate account details, handle transfer requests, and maintain transaction records.



Functional Requirements worth considering

5. **Account Alerts and Notifications:**

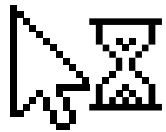
   - The system should provide the capability to send automated notifications and alerts to users for activities such as low balance, transaction confirmations, and security alerts.



Non-Functional Requirements worth considering

1. **Security:**

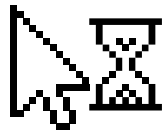
   - The system should adhere to industry-standard security measures to protect user data, such as encryption of sensitive information, secure communication protocols (e.g., HTTPS), and secure storage of passwords.



Non-Functional Requirements worth considering

2. **Performance and Scalability:**

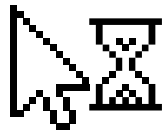
   - The system should be capable of handling a large number of concurrent users and transactions without significant performance degradation.
   - Response times should be within acceptable limits to ensure a smooth user experience.



Non-Functional Requirements worth considering

3. **Availability and Reliability:**

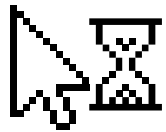
   - The system should be highly available and reliable, minimising downtime and ensuring continuous access for users.
   - Robust backup and disaster recovery mechanisms should be in place to safeguard against data loss and system failures.



Non-Functional Requirements worth considering

4. **Usability:**

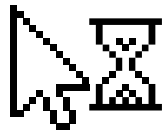
   - The user interface should be intuitive, user-friendly, and accessible, catering to a wide range of users, including those with disabilities.
   - The system should provide clear instructions, error messages, and appropriate feedback to guide users effectively.



Non-Functional Requirements worth considering

5. **Compliance and Regulations:**

   - The system should comply with relevant legal and regulatory requirements, such as data protection laws and financial industry standards (Bank of Ghana laws).
   - Audit trails and transaction logs should be maintained to support compliance and facilitate auditing processes.



Still to Come....

- System Architecture
- Performance
- Scalability
- Reliability
- Security