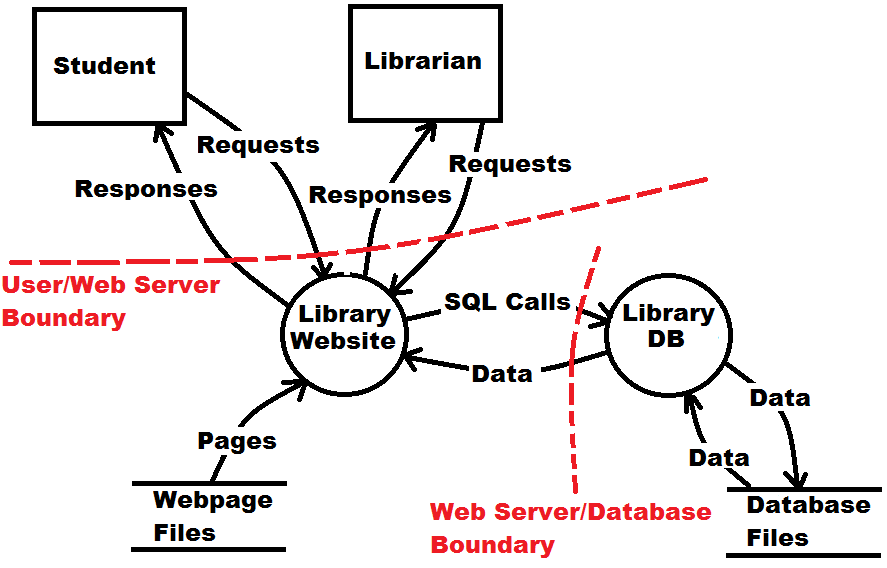
Exam 1 Review Software Engineering

**Security Issues in Software Engineering**

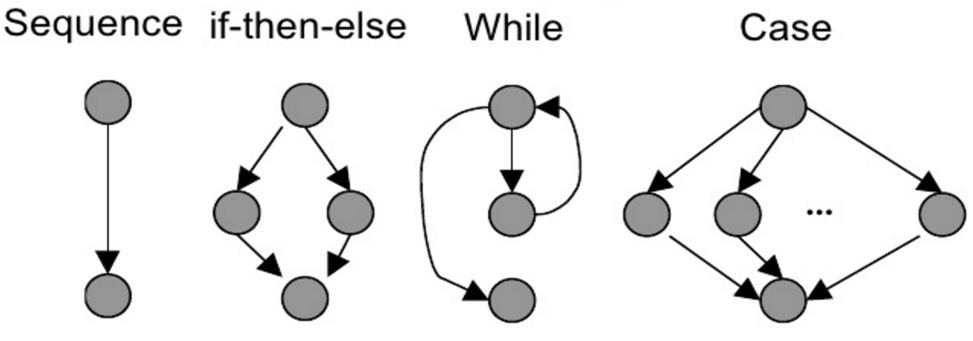
* **Software development security issues**
* One needs to develop security requirements. (Risk analysis and develop security polices)
* **Security threat models**



* **Security Architecture**
* **Designing with security in mind**
  + **Security components and techniques**
* Techniques:
  + Barriers: separate different parts of a complex system- require authentication to access certain systems or parts of the system. Isolate components (Ex: some computers are not connected to a network.) Firewalls (NAT routers act as a firewall for our groups Raspberry Pi.)
  + Authentication: establishes the identity of an agent (Ex. What does the agent know, posses, have physical access to? What are the physical properties of the agent? password, url, smart card, network, computer, finger print.)
  + Authorization: establishes what an authenticated agent may do (Ex. Access control lists and group memberships).
  + Encryption: Allows data to be stored and transmitted securely, even when the bits are viewed by unauthorized agents, and the encryption algorithms are known.
* **Programming secure software**
  + **Most dangerous errors**
  + SQL Injection- malicious users can inject SQL commands using web page input. (Attack Frequency-High. East of detection- Easy Consequences- Security bypass, data loss. Remediation Cost- Low.)
  + OS command injection: uses a web interface to execute OS commands on a web server. (Attack Frequency-Often. East of detection- Easy Consequences- Code execution, data loss. Remediation Cost- Medium.)
  + Buffer Overflow: a program overwrites memory adjacent to a buffer that should not have been modified. (Attack Frequency-Often. East of detection- Easy Consequences- Code execution, Denial of Service, data loss. Remediation Cost-Low.)
  + Cross-site scripting: malicious users inject Javascript or other client side content into a web page that your web server generates. (Attack Frequency-Often. East of detection- Easy Consequences- Code execution, Security Bypass. Remediation Cost-Low.)
  + Missing authentication for critical function: attack technique that targets critical functions or modules of software that does not perform security checks before accepting or processing input. (Attack Frequency-Sometimes. East of detection- Moderate Consequences- Security Bypass. Remediation Cost-Low to High.)

**Legacy Systems and Software Reuse**

* **Software component reuse**
* **Designing software systems for change**
  + **Modular/replaceable components**
* **Designing software systems for reuse**
  + Class Hierarchies
  + Inheritance and Abstract Classes.
* **Legacy code**



* + **Generating requirements, documentation, models, etc.**
  + **Modifying**
  + **Testing**
  + **Dependencies**

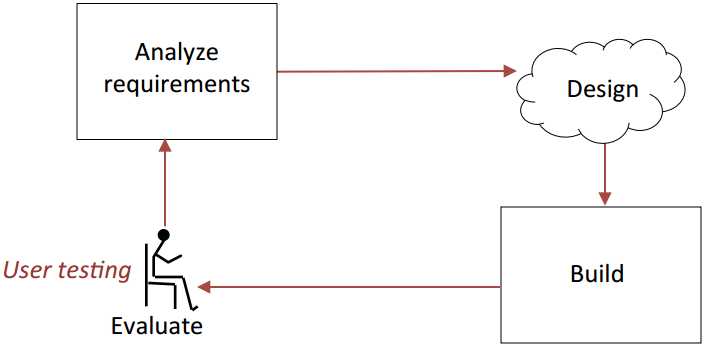
**Software System Performance 1**

* **When and how performance matters in computer systems**
  + Very large computations – Processing time may be measured in days.
  + Performance considerations- Inspect software algorithms for efficiency, consider data pipeline. Split input data into chunks.
  + User Interfaces—Where humans have high expectations
  + Performance considerations- start-up time factors for applications, web interfaces must process requests in near real time. User client-side and server side programs to increase performance.
* **High performance computing**
  + **HPC vs HTC**
  + High Performance computing: Tightly coupled nodes communicating with low latency, systems enable users to run a single instance of parallel software over many nodes.
  + High Throughput Computing: use of many heterogeneous computing resources over long periods of time to complete a computational task. Many times, HTC systems are created with unreliable components. Schedules give work to nodes in the HTC system, and eventually the work is completed.
* **Performance challenges for software systems**
  + **Bottlenecks**
  + Hardware bottlenecks
  + Inefficient software.
  + CPU performance sometimes.
  + **Timescales of components**
  + **Utilization of different components**
* **Predicting system performance**
  + **Simulations**
  + **Direct measurement**
  + File I/O
  + CPU
  + Network
* **Fixing poor performance**
* **Software performance requirements documentation**

**Software System Performance 2**

* **Analysing system performance**
  + Product wasn’t developed using performance testing requirements.
  + When a problem develops or is found, no one takes responsibility.
  + Developers don’t use or don’t know about tools that are available to solve the problem.
  + After developing a list of possible causes, there’s no elimination of the major problems.
  + Often developers don’t have the patience to examine the large amounts of data generated by performance testing.
* **Performance engineering motivation**
* **Performance metrics**
  + Reliability- A system X always outperforms a system Y if the performance metrics indicates that X always outperforms Y.
  + Repeatability- The same value of the metric is measured each time the same experiment is performed.
  + Consistency- Units of the metrics are the same across different systems and different configurations of the same system.
  + Ease of measurement- If a metric is hard to measure, it is unlikely anyone will use it.
  + Independence- metrics shouldn’t be defined to favor certain systems.
  + Common Performance Metrics- Function/system throughput:
* **Performance engineering documentation**
  + **Requirements**
  + **Functional specifications**
  + **Design specifications**

**Usability and User Interfaces**



* **Importance and development of user interfaces**
  + **Mock up, wire frames, focus groups**
  + A group of people assembled to participate in a discussion about the software product. 3 to 5 users peer group meeting with similar characteristics. Structured set of questions to be repeated with multiple user groups.
* **Usability requirements**
  + **Accessibility:** People may have disabilities (ie. poor eyesight, lack of hearing, poor annual dexterity) and people who may have a limited knowledge of English.
  + Support Users with disabilities are likely to arise in the user interface
  + You may have a legal requirement to support people with disabilities.
  + **Equipment**
  + Your project may require you to support computers/systems with poor performance, limited screen sizes, bad network connections.
  + Performance user testing with both good and bad equipment.
* **Mental models**
* **Interface functionality**
* **UI design**
  + **GUI-** What you see is what you get.
  + Can be intuitive and easy to learn
  + Users get immediate feedback
  + Icons can be language-independent
  + Not suitable for some complex interactions
  + Difficult to build scripts
  + Only suitable for human users
  + **Command line**
  + Allows complex instructions to be given to the computer
  + Skilled users can input commands quickly
  + Unless very simple, requires learning or training
  + Can use multiple programming languages
  + Suitable for scripting
  + **Help systems**
  + Must prototype with mixed users.
  + Must have many routes to same information
  + Categories of help:
    - Overview and general information.
    - Specific or context information
    - Tutorials (general)
    - Wizards
    - Emergency (“I am in trouble…”)

Usability and User Interface 2

* **Four computing eras**

1960s: Mainframe Era One computer per many users (ex. UNIVAC, NCR, RCA)

1980s: Personal Computer Era One computer per user (Lenovo, Apple, HP, Dell)

2000s: Mobility Era Several computers per user(Apple IOS, Google Android devices, Blackberry)

2020 and beyond: Ubiquity Era Thousands of computers per user (Cloud resource provisioning, Amazon AWS Lambda, Smart homes, Internet of things)

* **HCI architectures**
  + **Unimodal-** A system that is based on a single interface mode is called unimodal. (Audio sensor, visual)
  + **Multimodal**
* **Usability goals**
  + **Satisfaction**
    - **Effective functionality, nice experience**
  + **Robustness**
    - **Stable, low probability of (system) interface errors**
  + **Efficiency**
    - **Good use of user’s time, system resources**
  + **Learnability**
    - **Low learning cure to use UI**
  + **Memorability**
    - **User remains efficient even though the UI is used infrequently**
* **Usability design**
  + **Users with disabilities**
  + **Elderly users**
  + **Children**

**Software Reliability**

* **Reliability terminology**
  + **Failure, fault, error, fault avoidance/detection/tolerance**
  + Error: human action that results in software containing a fault.
  + Fault avoidance: Systems have the objective of presenting fault-free (bug-free) software.
  + Fault detection (testing and verification): detect faults (bugs) before the system is put into operation or when discovered after release.
  + Fault tolerance: build systems that continue to operate when problems (bugs, overloads, bad data, etc.) occur.
  + Failure: any observable divergence of software behavior from user need/requirements.
  + Failure intensity: the number of failures per time unit. This is a way of expressing reliability.
* **Software vs. hardware reliability**

|  |  |  |  |
| --- | --- | --- | --- |
| **Software Reliability** | **Hardware Reliability** | | |
| Failures are primarily due to design faults. Repairs are made by modifying the code. | | Failures are caused by deficiencies in design, production, and maintenance. |
| No “wear-out” phenomena. Errors can occur without warning. | Failures are caused by wear or energy/environment attributes. Sometimes a warning is available before a failure occurs. | | |
| No equivalent preventive maintenance for software | Repairs can be made that makes hardware more reliable. | | |
| Reliability is not time dependent. Failures occur when the logic path the program takes contains an error. | Reliability is time related. Failure rates can be decreasing, constant, or increasing with respect to operating time. | | |
| External environment conditions do not affect software reliability. Internal conditions, such as insufficient memory or inappropriate clock speeds do affect software reliability. | Reliability is related to environmental conditions. | | |
| Reliability can’t be predicated from knowledge of design, usage, or stress factors. | Reliability can, theoretically, be predicted from design factors and physical attributes. | | |
| Reliability can’t be improved through redundancy of software. Redundancy will simply replicate the same error. | Reliability can usually be improved through redundant hardware. | | |
| Failure rates of software components are not predictable. | Failure rates of hardware components are somewhat predictable according to known usage patterns. | | |

* **Software reliability modelling**
  + **Limit the visibility of information in a program**
  + **Check all inputs for validity**
  + **Provide a handler for all exceptions**
  + **Minimize the use of error-prone constructs**
  + **Provide restart capabilities**
  + **Check array bounds**
  + **Include timeouts when calling external components**
  + **Name all constants that represent real-world values**
* **Software reliability metrics**
  + **Mean time between failures (MTBR)**
  + **Mean time to failure (MTTF)**
  + **Availability (up time)**
  + **Mean time to repair (MTTR)**
  + **Market measures** 
    - **Complaints**
    - **Customer retention**

**Software Testing Strategies 1**

* **Testing strategies**
  + **General characteristics**
  + **Verification and validation**
* **Levels of testing**
  + **Unit testing**
    - Interface, data structurers, boundaries, basis paths, error handling
  + **Integration testing**
    - Non-incremental, incremental
  + **Validation testing**
    - Based on requirements
  + **System testing**
  + **Regression testing**
  + **Smoke testing**
* **Error discovery and handling**
* **Debugging**

**Software Testing Strategies 2**

* **Static verification testing**
  + **Program inspections**
    - **Data, control, I/O, interface, storage, exceptions**
* **Dynamic verification testing**
  + **Interface**
    - **Parameters, shared memory, message passing protocols**
  + **Unit testing: Basis paths** 
    - **Flow graphs, cyclomatic complexity, basis set, test cases**
  + **Integration testing**
  + **Performance/stress testing**
  + **Acceptance testing**

**Acceptance Testing**

* **Acceptance Criteria**
* **Acceptance Testing Approach**
  + **Iterative/Incremental/Sequential**
* **Acceptance Test Reports/Documentation**
* **Delivery of Software**