CS155-2 Module 1 Reviewer

Software testing

The three stages of testing

- **Development testing**, system is tested during early development for bugs.
- **Release testing**, a seperate testing team tests the complete and final product before selling.
- **User testing**, user tests the system in their own enviroment. One example is **acceptance testing**

Development testing

Three stages within development testing

- **Unit testing**, testing each individual component. Like testing functions on their own
- **Component testing**, several individual units are integrated as one small system.
- **System testing**, some or all of the system is integrated together and fully tested as a complete package.

Unit testing

Three parts

- **Setup**, variables and context are setup
- **Call**, the function is called and executed
- **Assertion**, the result of the function called is compared against the result in the test cases.

Software Evolution

Phaseout

Eventually, all software becomes legacy software once it becomes to expensive to maintain and there is no need for it.  Alt text

Legacy Software

Legacy systems are older systems that rely on languages and technology that are no longer used for new systems development.

The structure and quality may likely have degraded over the course of its updates. It is also likely compatible with older hardware best.

Issues surrounding legacy software

Risks in replacing legacy systems

- **Missing/incomplete specification**, becomes harder to recreate the functionality of the old legacy sooftware.
- **Embedded business rules**, sometimes business processes are faciliated by software functionalities instead of formal documentation. *For example, an insurance company may have embedded its rules for assessing the risk of a policy application in its software.*
- **General development risk**, making any software in general is risky, as the deadlines may not be met, or something goes wrong along the way and the project forces to discontinue.

Problems in reengineering legacy software

- **Outdated programming convention**
- **Obsolete technology**
- **Outdated system documentation**

- **Bad structure from updates**, this will make it harder for programmers to understand the inner workings, costing more time and thus money.
- **Dependency on old hardware**, the system may have optimization and hacks that work on older hardware only.
- **Different data structure**, although the old and new software mean to store the same information. The structure may be different and will need conversion, or cleaning for invalid and empty data.

What to do with legacy software

Analyzing the state of the software

Legacy software has 2 aspects we need to worry about, **value** and **quality**. **Value** is how useful the software still is to the business, **quality** refers to the code of the software, high quality means its easy to understand code and maintain the software, in other words, quality is **cost**.

Strategies to legacy software

Quality	Value	Strategy
Low	Low	Scrap the legacy system
Low	High	Consider reengineering to improve quality, or replace with suitable off the shelf software
High	Low	Continue normal updates, stop and scrap once it becomes expensive to update
High	High	Continue updates as normal.

Software maintaineance

Software maintenance is the general process of changing a system after it has been delivered.

Types of software maintaineance

- **Fault repairs**, fix bugs and glitches
- **Enviromental adaption**, Adapt to new platforms, hardware, scale, etc.
- **Functionality addition**, add new features to meet new reuquirements

Why adding functionality is difficult

Experience has shown that it is usually more expensive to add new features to a system during maintenance than it is to implement the same features during initial development

- **A new team has to understand the program being maintained.**
- **The development team are often not incentivized to write maintainable software**. Usually the development team and the maintaineance team are seperate, the devs dont gain much from writing

maintainable software as they wont continue work on it.

- **Maintaineance is unpopular**, it is perceived as less skillful, so often inexperienced programmers get this job.
- **Maintained software degrades with every update**, after changes, programs become harder to understand and change. Old documentation can be inconsistent, incomplete or even just lost.

The solution? Hire the same programmers who developed the software to also maintain it.

Reengineering software

Software reengineering is concerned with restructuring and redocumenting software to make it easier to understand and change.

Activities in reengineering

- **Source code translation**, convert into a modern programming language or version of it.
- **Reverse engineering**
- **Structure improvement**, the control structure is analyzed to become more efficient and readable.
- **Modularization**, logically related parts of the software is grouped, redundancy is removed.
- **Data reengineering**, redefining database schemas, cleaning duplciates, fixing invalid data.

Refactoring

Refactoring, making small program changes that preserve functionality, can be thought of as preventative maintenance.

Activities in refactoring

- **Removing duplcicate code**
- **Shortening long methods**
- **Reducing uneeded switch cases**
- **Data clumping**, often many of the same data items are used again and again throughout the program. Just turn it into one class.
- **Speculative generality**, This occurs when developers include generality in a program in case it is required in the future.

Dependable software

About system failures

Importance of avoiding system failures

- System failures on bnig systems affect many
- Users dont want to use any unrelaible software

- Some costs can be grave, like nuclear reactor control software, airplane autopilot, power grid conttrol. The costs dont only affect profit, but lives.
- Undependable systems may cause data loss, data is expensive to collect and maintain, sometimes more expensive than the servers storing it. *(Ask facebook when they sold to cambridge analytica lol)*

Types of failures

- **Hardware failure**, something may go wrong in the hardware, manufacturing errors, too hot or too damp, or components are just old.
- **Software failrue**, fail in specification, design, implementation
- **Operational failure**, users fail to operate the system as intended by the designers. UI/UX is important.

Dependable systems

Measures of dependability

- **Availability**, the availability of the system to be running and serve users
- **Reliability**, the system being able to serve as expected of users.
  - Avoid the introduction of accidental errors
  - Design verification and validation processes to find errors
  - Configure and deploy the system according to its hardware enviroment.
- **Safety**, how likely the system can cause damage to users and its enviroment.
- **Security**, how well internal mechanism and user data can be protected from external attacks
  - Design protection mechanisms that guard against external attacks. -Include system capabilities to recognize external cyberattacks and to resist these attacks.
- **Resilience**, how well the system can maintain critical services and get back on its feet when something wrong happens. Whether its an error, or a cyberattack.
  - You design the system to be fault tolerant so that it can continue working when things go wrong.
  - You design systems so that they can quickly recover from system failures and cyberattacks without the loss of critical data.

Noteowrthy properties

- **Repairability**, how easy it is to diagnose the problem and fix it.
- **Maintainability**, how easy it is to add functionality to support new requirements. This means readable code and good documentation to save time trying to understand the code.
- **Error tolerance**, User errors should be detected and fixed automatically.

Reliability engineering

About Errors

Types of errors

- **Human error**, Human behavior that results in the introduction of faults into a system
- **System fault**, a characteristic of the software that leads to system error.
- **System error**, a state that can lead to unexpected system behaviour
- **System failure**, The system fails to serve users.

> Faults dont immediately mean failure, erroneous code may never be executed afterall.
>
> Same goes with errors, systems may have protection facilities against errors.

Fault management

- **Fault avoidance**, use processes and methodologies that help avoid design errors and introduce less errors in the system in the first place.
- **Fault detection and correction**, Verification and validation processes are designed to discover and remove faults in a program. Like testing routines, etc.
- **Fault tolerance**, The system is designed so that faults or unexpected system behavior that occur at runtime do not lead to system failure. Systems can have run-time systems to detect and try to automatically resolve errors.

Applying all these **cost money**, funny thing is cost rises as more errors are found.  Alt text

As you find more errors, it takes longer to find obscure errors. Software companies eventually accept that their software will always contain some residual faults. Sometimes its cheaper to let it fail then to try an fix it.

Reliability requirements

Types of requirements

- **Functional requirements**, goals related to expected features in the product as well as expected behaviour
- **Non-functional requirements**, goals unrelated to the features but to the executing abiltiy of the system.(*Eg: Performance, memory, etc.*)

Quantitative markers for reliability

- **Probablity of failure on demand (POFOD)**, the probability that the system will fail on a request. A POFOD of 0.001 means there is a 1/1000 chance the system will fail on a request
- **Rate of occurance of failures (ROCOF)**,
- **Availability**, the probability the system will remain operational on a request. An **AVAIL** of 0.998 means in 1000 time units, the system will run 998 time units operational.

> We use POFOD to measure if the failure will cause severe damages. Like a nuclear control system, or hospital equipment, etc.
>
> We use ROCOF when requests are constantly being made, like a social media site as users from all around the globe keep posting
>
> AVAIL is relevant in continous systems, like railway signalling, phone switching, clock apps.

Statistical/Reliability testing

Reliability testing (Statistical testing) is running the program and determining if the required level of reliability is achieved. This is testing software for reliability rather than fault detection.

Measuring the number of errors allows the reliability of the software to be predicted

Problems

- Operational profile may not be accurate
- High cost to generate test dataset
- Need high number of failures to compute failure, but reliable systems will rarely fail
- Recognizing failure is not always apparent. We may miss it.

Operational profile

This is a set of test data whose frequency matches the actual frequency of these inputs from 'normal' usage of the system

It can be generated from real data collected from an existing system or (more often) depends on assumptions made about the pattern of usage of a system.

Why do we need that?

- Investors can understand that engineering high levels or relibability cost more
- It provides a basis for assessing when to stop testing a system
- Its how you can decide on different stratgies to improve the reliability of the system
- It is evidence to pass regulations for critical systems. (*Eg: Regulations require flight system applications to operate with low latency*)

Guidelines for specifying reliability requirement

- Specify the availability and reliability requirements for different types of failures. Severe failures should be lower than trivial errors.
- Consider how much reliability is needed for the system. Simple apps like games, etc can get away with a few bugs. But critical apps like flight control, nuclear reactor control must be close to absolute perfection.
-

Extra useless stuff

Self monitoring architectures

Multichannel architectures that run the same computation in parallel, if the results are different the system might be faulty.

N-Version programming

An odd number of computers, or 3, carry out computations. The results are compared in a voting system and the majority is assumed to be the result.

Guidlines for dependable programming

1. Limit the visibility of information in a program, using java's private and public, using get() and set().
2. Check all inputs for validity
    - Size Checks, check input does not exceed max size
    - Range Checks, check input is within range
    - Representation Checks, check input doesnt contain invalid chars
    - Reasonableness checks, check if it is within extreme
3. Provide a handler for all exceptions
4. Minimize the use of error-prone constructs
    - Goto statements
    - Floating point numbers
    - Pointers
    - Dynamic memory
    - Stack overflow from recursion
    - Inheritance
    - Unbounded arrays
5. Provide restart capabilities
6. Check array bounds
7. Include timeouts when calling external components
8. Name all constants that represent real-world values

Safety Engineering

Types of safety-critical software

These software are ones with detrimental consequences should it fail.

- Primary safety critical software

These are software that directly results in enviromental damage, human harm, etc. An insulin pump is an example as should the software fail or be inaccurate, the patient may be underdosed or overdosed with insulin.

- Secondary safety critical software

This is software that does not directly result in damage. An example is computer aided modelling for some object, if the software is inaccurate, the built object may not work properly and cause damage.

Strategies against software faults

- Hazard avoidance

Implement steps to avoid hazard in the first place. For example a hydraulic press should be designed to start when 2 buttons are pushed, needing both hands of the operator.

There was an accident where a woman using the single button press accidentally triggered the machine while her arm was still under the press, resulting in her arm being flat.

- Hazard detection and removal

Hazards are detected and removed. For example in a chemical plant, if the pressure is too much, a relief valve is opened to reduce the pressure before an explosion.

- Damage mitigation

When you cant stop hazards, best you can do is reduce the damages. An aircraft engine has automatic fire extinguighers. It doesnt stop fires in the first place, but its does reduce the damage the fires will cost.

Terminologies

Term	Meaning
Accident	An unforseen and unplanned incident resulting in damage or injury or death
Damage	The measure of loss from an incident, could be monetary, injuries, deaths.
Hazard	A condition with potential leading into an accident. Like faulty insulin pumps, the person suffering afterwards is the accident
Hazard probability	The probability of the events occurring which create a hazard
Hazard severity	An assessment of the worst possible damage that could result from a particular hazard. A hazard that leads to the accident of many deaths is "catastrophic"
Risk	A measure of probability that a system will cause an accident and the impact. The hazard probability, the hazard severity are considered.

Hazard-driven safety specification

The 4 steps in hazard identification

- Hazard identification

The hazard identification process identifies hazards that may threaten the system

- Hazard assessment

The hazard assessment process decides which hazards are the most dangerous and/or the most likely to occur. So that they may be prioritized accordingly

- Hazard analysis

An analysis to find the possible root causes that leads to a hazard

- Risk reduction

A process that should the hazard occur, find ways to mitigate the damage.

Assessing hazards

Hazards are typically categorized into 3 categories

- Intolerable risk

These are hazards that may cause human lives, systems are designed as much as possible to not cause a hazard, and if the do, detect and stop

- ALARP, as low as reasonably practical

These are risks where the impact is not as tragic as death, but still pretty bad, and the chance of occuring is low. The system is designed so the chance stays low and is reasonably cost efficient.

- Acceptable risks

These are risks where the accidents only result in minor damage. designers should still try to lower these risks, but having these risks dont increase the cost much.

- Negligeble risk

As the name suggest, thi is a risk we can simply not care about.

Below are some practicaly examples on categorizing hazards.

Hazard	Probability	Impact	Risk	Acceptability
Insulin overdose computation	Medium	High	High	Intolerable
Smart watch battery failure	Low	Low	Low	Aacceptable
Faulty car autopilot	Medium	High	High	Intolerable
Faulty automatic door sensor	Medium	Low	Low	ALARP

Hazard analysis

Hazard analysis is the process of discovering the root causes of hazards in a safety-critical system. Your aim is to find out what events or combination of events could cause a system failure that results in a hazard.

Typically you go either a top-down or a bottom-up approach. So in a top-down approach, you start with the accident, find out what may cause it, and so on.

Risk reduction

See Strategies against software faults

Safety engineering process

In order to pass regulators, your system or software needs evidence that its safe.

- a full system specification with records of safety checks

- Evidence and results of the verification and validation processes that have been carried out
- Evidence that the organizations developing the system have defined safety assurance reviews. There must also be records showing that these processes have been properly enacted

Safety assurance processes

- Hazard analysis and monitoring,

Hazards are traced from the analysis all the way through system valdiation

- Safety reviews
- Safety certification

Safety critical components are formally certified by a third party group deciding if its considered safe for use.

For example the IEE(International Elevator and Equipment) provides certification to allow buildings to facilitate their elevators. Additionally, they also do routine inspections.

Individual responsibility

Individuals who have safety responsibilities should be explicitly identified in the hazard register.

This is so that

- When people are identified, they can be hold accoutnable
- In the event of an accident, there may be legal action and thus it should be possible to identify the responsible.

Formal verification

Secuirty Engineering

Compromisable aspects

These are things that you can lose when someone tries to attack you.

- Confidentiality, keeping information a secret
- Integrity, knowing your data is uncorrupt and true. An attacker may delete and alter data, making you question if the data is true or not.
- Availability, the ability to simply operate. Attackers like to enact DDoS attacks to shut down services.

3 Levels of secuirty

- Infrastructure secuirty, the secuirty of all systems and networks to provide infrastructure and services to an organization
- Application security, The secuirty of individual applications
- Operational security, secure operation of the use of the business's systems.

Infrastructure security

Infrastructure security is primarily a system management problem, where system managers configure the infrastructure to resist attacks

1. User and permission management involves adding and removing users from the system, ensuring that appropriate user authentication mechanisms are in place, and setting up the permissions in the system so that users only have access to the resources they need.
2. System software deployment and maintenance involves installing system software and middleware and configuring these properly so that security vulnerabilities are avoided. It also involves updating this software regularly with new versions or patches, which repair security problems that have been discovered. Operating System Generic, shared applications (browsers, email, etc.) Database management Middleware Reusable components and libraries Application Network Computer hardware
3. Attack monitoring, detection, and recovery involves monitoring the system for unauthorized access, detecting and putting in place strategies for resisting attacks, and organizing backups of programs and data so that normal operation can be resumed after an external attack.

Security and dependability

Term	Definition
Asset	Something to be protected (Eg: Data)
Attack	An exploitation of vulnerability where the attacker tries to compromise something (Eg: Privilege escalation through weird action in OS)
Control	A protective measure to reduce vulnerabilities (Eg: Escaped input to prevent XSS or SQL Injection)
Exposure	Possible loss or harm to the system (Eg: Finanical issues from legal action from users affected by hack)
Threat	Circumstances that have potential to cause harm to the computer (Eg: A hacker guessing credentials)
Vulnerability	A weakness in the system that can be exploited (Eg: Unhashed passwords 💀)

Control methods

- Vulnerability avoidance

Controls that are intended to ensure that attacks are unsuccessful. (Eg: Not connecting to the internet)

- Attack detection and neutralization

Controls that detect and repel attacks. Usually the monitor operation and notice unusual usage patterns. They can shut down parts of the system to prevent attacks from continuing.

- Exposure limitation and recovery

Controls that help support recovery and mitigate loss. Backing up the data occasionally.

Making a documentation on security policy

- The assets to be protected, not all assests need to be protected, some are just fine letting anyone view.
- The level of protection for assets, more critical assets like data should have stronger security. Otherwise costs will incur!
- The responsibilities of individual users, managers, and the organization. Eg: Users should think of passwords stronger than their brithdate.
- Existing security technologies and procedures that should be maintained

Assessing security risks

- Preliminary risk assessment

In here we identify generic risks that the system may have and decide on the feasible solutions to provide adequate security for reasonable cost.

- Design risk assessment

This assessment takes place during the system development, the results may change the security requirements and possibly new requirements.

- Operational risk assessment

This assessment focuses when the system is beign sued and the risks that can occur during operation. For example, users who leave a computer unattended with their account session ongoing. To solve this, simply add a simple timeout to automatically log out the users after some inactivity.

Security requirements

The preliminary risk assessment generally needs these requirements for support:

- Risk avoidance requirements
- Risk detection requirements
- Risk mitigation requirements

Risk-driven security requirements process

1. **Asset identification**, find out what assets need protection
2. **Asset value assessment**, find out how important an asset is
3. **Exposure assessment**, find out how much damage will occur if the asset is compromised
4. **Threat identification**, find out possible threats to the system
5. **Attack assessment**, decompose the threats into potential attacks that will occur on the system
6. **Control identification**, propose the solutions to mitigate or prevent the attack
7. **Feasibility assessment**, find out a cost-effective method to apply the controls
8. **Security requirements definition**, everything previously studied is compiled into the defined document.

Below is an example

Asset analysis

Asset	value	Exposure
Patient database	High, required for all consultations	High, finanical loss from legal, recovery, reduction of business, reputation, etc.
Single patient data	Low, but high for important people	Low, possibly ruined reputation

Threat analysis

Threat	Chance	Control	Feasibility
unauthorized user gains access	Low	Only allow system management from specific locations that are physically secure	Low cost, risk at key distribution

Misuse case

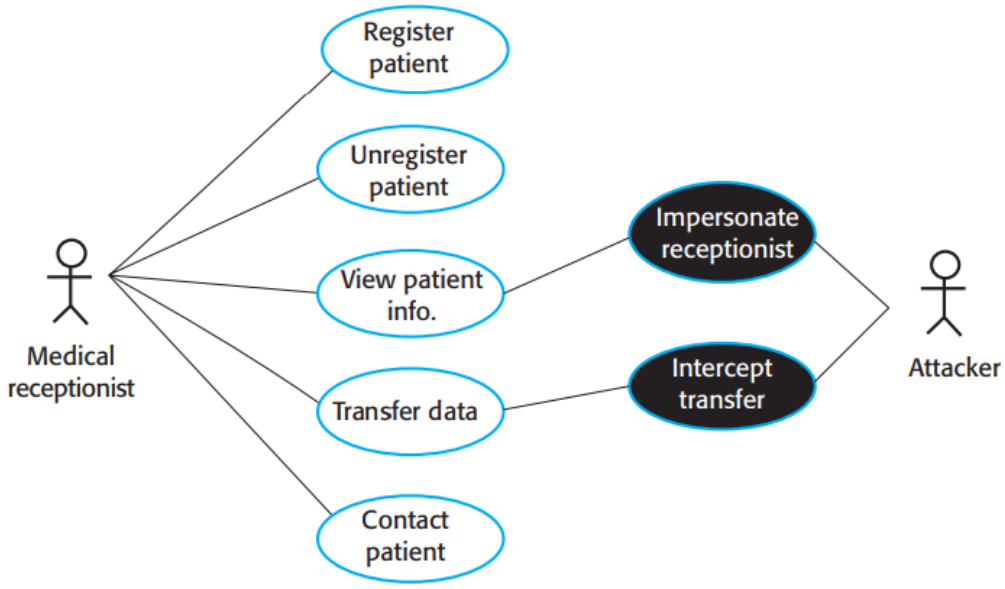

Figure 13.8 Misuse cases

Remember UML use case diagrams, these can also represent how an attacker might misuse a system feature to cause trouble.

Architecture design for security

- **Protection**, how should one organize the system components and assets so that they can be protected from an attack
- **Distribution**, how should one organize the system and assets so losses from an attack are minimized

System layers

- Platform-level protection. The top level controls access to the platform on which the patient record system runs.
- Application-level protection, this level is built into the applciation itself. Autehnticating the user, authorizing the user.

- Record-level protection, this is relevant when data is directly accessed. For example checking if a user is authenticated and authorized to carry out operations on a record.

Design guidelines

1. Base security decisions on an explicit security policy

Policies define the “what” of security rather than the “how”.

2. Use defense in depth

you should not rely on a single mechanism to ensure security. Use multiple techniques and technologies.

3. Fail securely

Its probably inevitable that your systm will at some point fail. But it should not be a oppurtunity for hackers.

4. Balance security and usability

The demands of security and usability are often contradictory. To increase security, sometimes it can be at the cost of user experience.

Eg: Password requirements, need numbers, symbols, capitalized letters, etc.

5. Log user actions

One of the principles in AAA is accounting, its keeping track of everything that happens.

6. Use redundancy and diversity to reduce risk

Redundancy means to support more than one version of the software. Diversity means for the same software to be implemented on different platforms.

Eg: Having a mobile app that offers updates, but doesnt make only the lastest version able to work. The app also works on android, iOS, etc.

7. Specify the format of system inputs

8. Compartmentalize your assets

Dont let users access all data, follow the "need to knoww" principle.

9. Design for deployment

a development enviroment is different from production.

10. Design for recovery

Secure systems programming

//todo

You have reached the end