NAME:ADELAIDE SITSOFE AMA BOSSO

CSD 5.5

SOFTWARE FAILURES

1. **FBI Virtual Case File**

The US Federal Bureau of Investigation has often been criticized for not sharing leads between agents and divisions. Just before the 2001 terrorist attacks, the FBI hired Science Applications International Corp (SAIC) to develop Virtual Case File software (VCF). VCF was designed to manage case files electronically, so that any agent with suitable permissions can find relevant information. Originally scheduled for completion in 2003.

After repeated delays, a version was delivered in December 2004, but:

• Was about one tenth of originally promised
• Was eventually scrapped altogether
• Does not approach functionality of existing commercial packages
• Used as an extremely expensive prototype
• About $170 million wasted

Problems:

- Didn't meet specification.
- Wasn't delivered on time.
- It cost more than expected.

2.**Customer Database System**

In 1996 a US consumer group embarked on an 18-month, $1 million project to replace its customer database. The new system was delivered on time but didn't work as promised, handling routine transactions smoothly but tripping over more complex ones. Within three weeks the database was shut down, transactions were processed by hand and a new team was brought in to rebuild the system.

Problems:

• The design team was over-optimistic in agreeing to requirements

• Developers became fixated on deadlines, ignoring errors

3.**Payroll system**

The night before the launch of a new payroll system in a major US health-care organization, project managers hit problems. During a sample run, the off-the-shelf package began producing cheques for negative amounts, for sums larger than the top executive's annual take-home pay, etc. Payroll was delivered on time for most employees but the incident damaged the relationship between information systems and the payroll and finance departments, and the programming manager resigned in disgrace.

Problems:

• The new system had not been tested under realistic conditions

• Differences between old and new systems had not been explained (so $8.0 per hour was entered as $800 per hour)

GHANA

1.The enterprise computer system that was recently instituted in Ghana for education failed to achieve the expected result for some reason but it seems that many people have different views of why the system failed. The system was successfully deployed in production yet a core function failed to achieve the expected outcome, leading to panic in the stakeholder community across the country.A core function of the system was to assign students in various districts to schools in or near their districts, but upon deployment the system assigned many children to schools far from their homes with some up to 100 miles away. This led to utter confusion, and some disorder, amongst parents and children across the country.

Problems

- Didn’t meet specification
- It wasn’t maintainable

2.SOFTtribe

> According to the Managing Director of GWCL, Mr Clifford Braimah, the activities of the software company has become a drain to the resources of the company as such the earlier the contract is terminated the better.
>
> He explained that as part of efforts of GWCL to improve their customer service delivery they had contract with SOFTtribe in 2016 after the company introduced their new software dubbed “MX-Platform.”

They worked together to develop tools for meter reading, customer application and hosted them on SOFTtribe platform, they were responsible for managing the systems updates and database engine.

He observed that after working with them for two years his outfit began to receive notice of resignation from the employees of SOFTtribe which led the utility company to experience system challenges.

Mr Briamah noted that the replacements of workers made by SOFTtribe could not work up to expectation and they supplied bills which were inconsistent with the bills generated by GWCL.“Balance statement of accounts was different from water bills making it difficult to handle customers’ queries, E-Manager that allows managers to monitor the performance of meter readers and other allied staff started experiencing data compatibility issues,” he bemoaned.

In addition, Mr. Braimah said Disconnection and Reconnection application could not be developed, thus could not deliver the call center management system.

He noted that the problem started to affect revenue collection from 67 million a month, stressing that “we dropped to 60 million and below.”

“Their Chief Executive Officer (CEO) told us to purchase Point of Sale (POS) devices to allow all our cashiers in the country to be hooked on the system. We spent GH₵700,000 to procure the devices yet they could not fix it,” he said.

“Together we developed tools for meter reading, customer application and hosted them on SOFTtribe platform, they were responsible for managing the systems updates and database engine,” Mr Braimah said.

Problems:

- Product was not built to specification.
- There were too many bugs in the software.
- Extra money was spent in order to facilitate the fix.

3.Software breakdown at Ghanaian port

ICUMS was a new port clearing system that processes documents and payments through one window, a departure from the previous system where valuation and classification, risk management and payment were done by different entities.

The Integrated Customs Management System (ICUMS) introduced to accept the imports declarations, was not functioning for a period, leaving importers in the dark as to when they could clear their goods.For two days running, importers were disappointed as they failed to clear any goods. The new system has been hit with allegations of poor service delivery, lower revenue mobilization, among other criticisms. Also,the Daily Graphic reported that checks at the dedicated banks at the port in Tema that receive import revenues, revealed that no new payments were made during that period.

The system, which was previously run by the Ghana Community Network Services Limited (GCNet) and operated by West Blue Consulting, was expected to be replaced by the new system to ensure smooth clearance of goods.

Problems:

- Software didn’t meet specification
- It caused financial loss to the state

MYTHS SURROUNDING SOFTWARE ENGINEERING

Myth 1: You will learn everything you need to know at your job.

the technology landscape is constantly changing a, disrupting traditional ways of thinking. With that, many new technologies and methodologies have been introduced to keep up with this change. Does it mean technologies that your company is using will also be constantly changing and advancing? Yes and No. The truth is that not all companies have the capacity or appetite to stay at the forefront of the technology evolution. What this means is that you need to be responsible for your own learning, upskilling and staying competitive in your career.

Myth 2: Code geeks don't understand business needs.

This might have been the case in the infancy of software development, but today's programming professionals are highly trained and experienced and most likely have worked on many types of business models throughout their career. They've probably seen many types of problems before and solved them many times over. They may have operated within numerous companies or even run their own businesses. The software world is process-based and, ultimately, so is any efficient business. It follows a logical flow. A modern programmer is focused on getting machines to do the hard work for you, rather than fawning over new technology just for its own sake.

SOFTWARE DEVLOPMENT CYCLE MODELS

Software development life cycle (**SDLC**) is a series of phases that provide a common understanding of the software building process. How the software will be realized and developed from the business understanding and requirements elicitation phase to convert these business ideas and requirements into functions and features until its usage and operation to achieve the business needs. The good software engineer should have enough knowledge on how to choose the SDLC model based on the project context and the business requirements.

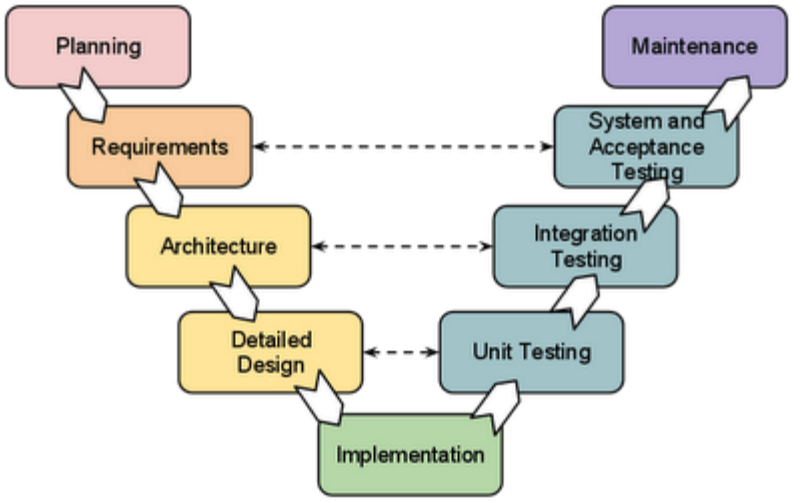
Therefore, it may be required to choose the right SDLC model according to the specific concerns and requirements of the project to ensure its success.

V-Shaped Model

Description

It is an extension of the waterfall model, Instead of moving down in a linear way, the process steps are bent upwards after the implementation and coding phase, to form the typical V shape. The major difference between the V-shaped model and waterfall model is the early test planning in the V-shaped model.

Advantages	Disadvantages
• Simple and easy to use • Each phase has specific deliverables. • Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle. • Works well for where requirements are easily understood. • Verification and validation of the product in the early stages of product development.	• Very inflexible, like the waterfall model. • Adjusting scope is difficult and expensive. • The software is developed during the implementation phase, so no early prototypes of the software are produced. • The model doesn't provide a clear path for problems found during testing phases. • Costly and required more time, in addition to a detailed plan



The usage

- Software requirements clearly defined and known

The big bang Model

The Big Bang model is an SDLC model where we do not follow any specific process. The development just starts with the required money and efforts as the input, and the output is the software developed which may or may not be as per customer requirement. This Big Bang Model does not follow a process/procedure and there is a very little planning required. Even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis.

Usually this model is followed for small projects where the development teams are very small.

Big Bang Model ─ Design and Application

The Big Bang Model comprises of focusing all the possible resources in the software development and coding, with very little or no planning. The requirements are understood and implemented as they come. Any changes required may or may not need to revamp the complete software.

This model is ideal for small projects with one or two developers working together and is also useful for academic or practice projects. It is an ideal model for the product where requirements are not well understood and the final release date is not given.

Big Bang Model - Pros and Cons

The advantage of this Big Bang Model is that it is very simple and requires very little or no planning. Easy to manage and no formal procedure are required.

However, the Big Bang Model is a very high risk model and changes in the requirements or misunderstood requirements may even lead to complete reversal or scraping of the project. It is ideal for repetitive or small projects with minimum risks.

The advantages of the Big Bang Model are as follows :

- This is a very simple model
- Little or no planning required
- Easy to manage
- Very few resources required
- Gives flexibility to developers
- It is a good learning aid for new comers or students.
-

The disadvantages of the Big Bang Model are as follows :

- Very High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Can turn out to be very expensive if requirements are misunderstood.

ARTIFACTS

An **artifact** is one of many kinds of tangible by-products produced during the development of software. Some artifacts (e.g., use cases, class diagrams, and other Unified Modeling Language (UML) models, requirements and design documents) help describe the function, architecture, and design of software. Other artifacts are concerned with the process of development itself—such as project plans, business cases, and risk assessments.

The term artifact in connection with software development is largely associated with specific development methods or processes e.g., Unified Process. This usage of the term may have originated with those methods

Artifact occasionally may refer to the released code (in the case of a code library) or released executable (in the case of a program) produced, but more commonly an artifact is the byproduct of software development rather than the product itself.

Much of what are considered artifacts is software documentation.

In end-user development an artifact is either an application or a complex data object that is created by an end-user without the

need to know a general programming language. Artifacts describe automated behavior or control sequences, such as database requests or grammar rules,[1] or user-generated content.

Artifacts are significant from a project management perspective as deliverables. The deliverables of a software project are likely to be the same as its artifacts with the addition of the software itself.

There are a lot of different parts of any given piece of software that can be artifacts. Here are a few of the most common examples:

Use Cases

Use cases are descriptions of how users are meant to perform tasks on a given program, website, or piece of software. These relate directly to the function of the site, making them important artifacts.

Unified Modeling Language (UML)

UML is a way of visualizing and plotting out the way a piece of software works. It works to map out links, processes, etc.

Like use cases, UML doesn't directly help software run, but it's a key step in designing and programming a piece of software.

Class Diagrams

Like UML, a class diagram is a way to map out the structure of a piece of software or application. Class diagrams are used to map out links and processes that happen between clicks in a more visual way.

Images

Any images used to help develop the piece of software are considered artifacts. These might be example images used to help in the design of the product, or preliminary design images. It could even be simple sketches and diagrams used to help map out the software.

Software Documents

The majority of the artifacts are software documents. Any document that describes the characteristics or attributes of a piece of software is an artifact. These can relate to the software's architecture, technical side, end-user processes, and marketing.

The majority of these artifacts won't even cross the mind of the user. They're meant for developers and those who might be interested in the software from a large-scale business perspective. Much of it is technical, and simply not of interest to the typical user.

Source Code

The source code is the language used to program a given piece of software. It's not the physical code itself, but the system that allows that code to work. This, too, is an artifact according to software developers.

Meeting Notes

Even meeting notes are artifacts in the world of software design. This may include full transcripts of meetings, or just jotted notes. Important design choices and aspects may have been made

during these meetings, making it important to include these in your repository.

Risk Assessments

A risk assessment offers a look at the potential risks and downfalls of a piece of software. It helps tell a developer what not to do and lists problems that the developer needs to find a way around. In some ways, these are some of the most crucial artifacts for developers to consider.

Prototypes

Any prototype of your program is an artifact. These might be fully-functioning pieces of the software or previews of certain parts of the program. Either way, they help a developer see what has been done and tried, and give them an idea of where to go next.

The Compiled Application

Once the piece of software is fully developed, it’s compiled into a usable application. This is the final artifact, and one of the only ones a typical user will care about. The compiled application will let the user install it onto their machine, and use it as its meant to be used.

There may be a number of these in the artifact repository. There could be different versions, from early prototypes to experimental builds and the final compilation.

The Types of Artifacts

There are three main categories that software artifacts fall under. These are code-related artifacts, project management artifacts, and documentation.

Code Artifacts

Every program is made up of codes, and each coded process produces artifacts. These include:

- Test Suites**: Coded test to run against the program in order to make sure a certain process is working**
- Setup Scripts**: Allow the program to run on a given machine**
- Complied Code**: The final, compiled code that goes out to the user**
- Test Output**: Logs and information that comes from the testing process**

Code artifacts are unavoidable, important by-products of the coding process. They let a developer test the ins and outs of their software, allowing them to perfect it for the user.

Documentation

Essentially, any piece of documentation relating to the software at hand is a relevant artifact. This includes:

- Diagrams: We talked about class diagrams above. Diagrams are a great way to visualize the internal processes of a given

program. These will be created throughout the coding process, particularly in the preliminary stages.

- End-User Agreements: An end-user agreement is any document meant for the user to read. This includes terms-of-service documents and anything that helps the user understand the program.
- Internal Documents: Any document that helps developers, bug-fixers, programmers improve on and understand the program. This also includes walkthroughs — guides that show testers, quality control people, etc. how to manage the application.

A lot of the artifacts in a given program fall under the documentation category. These are produced throughout the development process, from beginning to end. The more research that goes into a program, the more documentation artifacts there are. No matter how unimportant or preliminary the document may seem, these are important to keep track of.

Project Management Artifacts

These artifacts come about during the project management phase. They may come after routine tests of software, or after bug-checks. These artifacts relate to the ideal behavior of the program, and what the client wants of it.

They include:

- User Cases/Stories: These artifacts describe what the program should do in specific, technical terms. They detail what's wanted of the program, and how that might be achieved.

- Criteria: These artifacts layout the minimum acceptable requirements of the program/project. These typically come from the client or the project manager. This gives developers a concrete goal to work towards.

Project management artifacts let the development team know if they’re on the right track, and give them an idea of how to get there. These are crucial bits of feedback for everyone on the development team, new or old.