

**LIRA UNIVERSITY**

**FACULTY OF MANAGEMENT SCIENCES**

**DEPARTMENT OF COMPUTING AND INFORMATION SCIENCES**

**SOFTWARE EVOLUTION**

**COURSE UNIT:** SOFTWARE EVOLUTION

**COURSE CODE:** LCS 3202

**STUDENT’S NAME:**  TUMUKURATE DEBRAH

**REGISTRATION NUMBER:** 19/U/0025/LCS

**Qn.1 a) With examples discuss the Lehman’s laws of software evolution.**

**Ans:**

Law 1: *Continuing change* — A software system will become progressively less satisfying to its users over time, unless it is continually adapted to meet new needs. This implies that system maintenance is an inevitable process. As the system’s environment changes, new requirements emerge and the system must be modified. When the modified system is reintroduced to the environment, this promotes more environmental changes, so the evolution process starts again.

Law 2: *Increasing complexity* — A software system will become progressively more complex over time, unless explicit work is done to reduce its complexity. As a system is changed, its structure is degraded. The only way to avoid this happening is to invest in preventative maintenance. The company spend time improving the software structure without adding to its functionality.

Law 3: *Self-regulation* — The process of software evolution is self-regulating, with close to normal distribution of the product and process artifacts that are produced. System attributes such as size, time between releases, and the number of reported errors is approximately invariant for each system release. Lehman and Belady suggest that this law is a consequence of structural factors that influence and constrain system change, and organizational factors that affect the evolution process. The structural factors that affect the third law come from the complexity of large systems. As the company changes and extend a program, its structure tends to degrade. This degradation, if unchecked, makes it more and more difficult to make further changes to the program. Making small changes reduces the extent of structural degradation and so lessens the risks of causing serious system dependability problems. If the company try and make large changes, there is a high probability that these will introduce new faults. These then inhibit further program changes. The organizational factors that affect the third law reflect the fact that large systems are usually produced by large organizations. These companies have internal bureaucracies that set the change budget for each system and control the decision-making process. Such decisions take time to make and, sometimes, it takes longer to decide on the changes to be made than change implementation. The speed of the organization’s decision-making processes therefore governs the rate of change of the system.

Law 4: *Conservation of organizational stability* — The average effective global activity rate on an evolving software system does not change over time; that is, the amount of work that goes into each release is about the same. Over a program’s lifetime, its rate of development is approximately constant and independent of the resources devoted to system development. This law suggests that most large programming projects work in a ‘saturated’ state. That is, a change to resources or staffing has imperceptible effects on the long-term evolution of the system. This is consistent with the third law, which suggests that program evolution is largely independent of management decisions. This law confirms that large software development teams are often unproductive because communication overheads dominate the work of the team.

Law 5: *Conservation of familiarity* — The amount of new content in each successive release of a software system tends to stay constant or decrease over time. Over the lifetime of a system, the incremental change in each release is approximately constant. Adding new functionality to a system inevitably introduces new system faults. The more functionality added in each release, the more faults there will be. Therefore, a large increment in functionality in one system release means that this will have to be followed by a further release in which the new system faults are repaired. Relatively little new functionality should be included in this release. This law suggests that the company should not budget for large functionality increments in each release without taking into account the need for fault repair.

Law 6: *Continuing growth* — The amount of functionality in a software system will increase over time, in order to please its users.

Law 7: *Declining quality* — A software system will be perceived as declining in quality over time, unless its design is carefully maintained and adapted to new operational constraints.

The sixth and seventh laws are almost similar and essentially show that users of software will become increasingly unhappy with it unless it is maintained and new functionality is added to it.

Law 8: *Feedback System* — Successfully evolving a software system requires recognition that the development process is a multi-loop, multi-agent, multi-level feedback system; thus, for example, as a software system ages, it tends to become increasingly difficult to change due to the complexity of both the artifacts as well as the processes involved in effecting change. This law also implicitly recognizes the role of user feedback in providing impetus for future evolution.

**Qn.1 b). Discuss the importance of reverse and forward engineering as a tool for re-engineering.**

**Software Re-engineering** is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc.

**Forward Engineering:**

Forward Engineering is a method of creating or making an application with the help of the given requirements. Forward engineering is also known as Renovation and Reclamation. Forward engineering requires high proficiency skills. Forward engineering is a technique of creating high-level models or designs to make in complexities and low-level information. Forward Engineering applies of all the software engineering process which contains Software Development Life Cycle (SDLC) to recreate associate existing application. Forward engineering is the process of building from a high-level model or concept to build in complexities and lower-level details. This type of engineering has different principles in various software and database processes.

**Reverse Engineering:**

Reverse Engineering is also known as backward engineering, is the process of forward engineering in reverse. In this, the information is collected from the given or existing application. It takes less time than forward engineering to develop an application. In reverse engineering, the application is broken to extract knowledge or its architecture.  Reverse engineering plays a huge role in sparking innovative and productive minds that produce necessities across every industry. The reverse engineering process includes taking apart worn-down products to examine how individual parts work, and then incorporating past inventions into new ones.  It is the reorganizing and modifying existing software systems to make them more maintainable.

**Importance of Reverse Engineering**

1. **Exploring existing designs**

Reverse engineering allows us to see what already exists. This includes any parts, structures, or processes that could benefit communities in other ways. Examining current products leads to innovation and discovery, all thanks to reverse engineering.

1. **Reconstructing a product that is outdated**

A key part of redesigning an existing product is understanding the product itself. Reverse engineering provides the visual to work out outdated kinks in an older system. Quality is the most important aspect of this process.

1. **Discovering any product vulnerabilities**

Similar to the previous step, reverse engineering supports finding faults in the product. This is to ensure the safety and well-being of the product’s users. It is best for an issue to arise during the research phase rather than the distribution phase.

1. **Bringing less expensive and more efficient products to the market.**

Reverse engineering’s main goal is to lead engineers on a path towards innovation and success. Succeeding includes lowering manufacturing costs and raising product effectiveness as much as possible.

1. **Creating a reliable CAD model for future reference**

Most reverse engineering processes include a full-working CAD file for future references. A CAD file is created so the part can be examined digitally if future issues arise. This form of technology has enhanced engineering productivity and product expression.

1. **Inspiring creative minds with old ideas**

Lastly, reverse engineering gives way for innovative design. During the process, an engineer might discover a system that could be useful for a completely different project. This shows how engineering connects projects with previous knowledge.

**Differences between forward engineering and reverse engineering.**

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Forward Engineering** | **Reverse Engineering** |
| Basics | Forward engineering aims at re-developing an application with the provided resources according to the current requirements. | Reverse engineering aims at deducing the resources and requirements that went into developing an application in the first place. |
| Nature | It is perceptive in nature. | It is adaptive in nature. |
| Certainty | It always leads to the production of applications that implement the specified requirements. | Using this type of engineering, a user can easily yield a lot of ideas associated with the requirements of the implementation and development of the application. |
| Skills Needed | It requires high-proficiency skills. | It works even with a low level of expertise. |
| Total Time Required | It takes more time. | It takes comparatively less time. |
| Accuracy | The model in engineering must be complete and precise. | An inexact model can also yield partial information about the initial requirements and resources that went into its development. |
| Examples | A few examples of forward engineering are the construction of a DC motor, constructing kits (electronic), etc. | Performing research on various instruments is a commendable example of backward engineering. |

**Qn.2 a) Explain the various ways of evaluating legacy systems**

A legacy system is outdated computing software and/or hardware that is still in use. The system still meets the needs it was originally designed for, but doesn’t allow for growth. What a legacy system does now for the company is all it will ever do. A legacy system’s older technology won’t allow it to interact with newer systems.

A legacy system is an old or outdated system, technology or software application that continues to be used by an organization because it still performs the functions it was initially intended to do. Generally, legacy systems no longer have support and maintenance and they are limited in terms of growth. Legacy software can be described informally as old software that is still used to perform a useful job for an organization. Legacy software systems are comprised of programs that are still used by a business or have some potential value but were developed years ago using early versions of programming languages.

A legacy software application is an information system that may be based on outdated technologies, but is critical to day-to-day operations.

**Ways of evaluating legacy systems**

1. **Complete a Post-it Note Analysis**

**This includes the identification of the top three issues and the top three benefits of the system.** Factors to consider include:

* **Stability and Reliability**– Does the system stay live consistently or does the team experience downtime too frequently?
* **Maintainability** – Is the system easy to maintain? Can maintenance be done internally? Is the company that built the software still thriving? Is it offering support as the partner, or are they just another vendor? and willing to help?
* **Compatibility** – Is it possible to integrate new software to continually improve the system? Is it easy or difficult to connect with other systems?
* **Outdated Function** – Does the software still do everything the company need it to do? Is the team creating and using “hidden” manual workarounds?

1. **Estimate the Severity**

Legacy system drag has real business costs—both direct and indirect. The company need to put numbers on both. Follow this assessment:

* If the current system is often experiencing downtime, what opportunities are missed during that downtime?
* If the system provider no longer supports the technology, are there other vendors that can?
* How much time is being wasted?
* How much money is associated with that lost time or lost opportunity?
* How deeply is poor user experience contributing to staffing or customer attrition rates?

1. **Identify a Task Force**

Put together a team of internal and external users and stakeholders who would benefit most from legacy system optimization—or have the most to lose from not taking action. Start by identifying user types, then choose a representative from each group. Involve customers and/or partners and vendors where needed—the company can do that with a quick, simple email survey to assess ask what’s working, and what’s not.

1. **Complete a Deeper Assessment (SWOT)**

Complete a [SWOT analysis](https://www.wordstream.com/blog/ws/2017/12/20/swot-analysis). Each user type representative can work through a SWOT analysis for the parts of the system they use most. Below are a few questions from each category to aid in the system analysis.

* **Strengths** – What is currently working well in the system? What do the company like about the workflow that makes tasks easier? Does the system process things quickly?
* **Weaknesses** – Are there times when the company need to enter data into two or more different systems that don’t sync with one another? Does the current workflow leave too much room for human error by needing too many manual entries? Is there a lack of support from the current vendor?
* **Opportunities** – Can the company process more orders? Can the company improve the customer experience? Can the company save money by consolidating or integrating existing workflows?
* **Threats** – Will the company fall behind the competition? Does the system get in the way of exceptional customer service? Are employee onboarding and retention stressed by software or organizational shortcomings?

1. **Compare The Options to Make Informed Decisions**

Now that the company has outlined the issues, complied a team, and identified areas that require priority attention, the company is ready to weigh the options for action.

**What are the options?**

* Hire a consultant for a deeper analysis
* Make updates to the current system with the incumbent provider
* Implement integrated solutions to keep current functionality and improve productivity
* Start devising a plan to replace the system with a brand-new solution
* Review both off-the-shelf and custom solutions

**Questions to consider when determining the right decision for the company…**

* Do the company have or know a trustworthy development partner who understands the software?
* What is the cost difference between making updates and buying new? Consider integrations to mitigate costs and risks!
* Is the plan for updating going to tackle a symptom of an outdated system or the root cause?
* Do the company need to hire a consultant or get a business analysis to get a deeper assessment?

**Qn.2 b). What is the importance of legacy systems in software evolution?**

Some of the importance of legacy systems are:

* **Legacy systems are familiar**

Having a legacy system provides a unique feeling of familiarity and comfort. In actuality, this close attachment is among the most common reasons for retaining legacy systems. The entire workflow is smooth since everyone knows how to access records and use the software. As such, it makes it easier to carry out daily tasks with no hitches whatsoever.

* **Legacy systems are efficient for specific tasks**

Efficiency is another legacy software advantage. Some legacy systems, despite being old and outdated, still work perfectly fine. And most importantly, these systems are typically designed for maximum capacity and remain reliable and durable in most cases.

For example, banking systems use legacy systems and mainframes to facilitate most of their transactions. And although they might be difficult to maintain, they are still efficient in carrying out day-to-day operations.

* **Legacy systems ensure continuity in business operations**

Modernizing software or switching to entirely new technology is a delicate process that could put business operations on hold. Alternatively, maintaining the status quo (legacy systems) lets business operations run smoothly with no hitches or pauses.

* **Legacy system replacement is painful**

Tons of new technologies are currently taking over industries globally. However, switching from a legacy system to entirely new technology can be a resource-intensive task for any company. And when you acquire new software, you need to hire and train specialists to use it as well as consider all associated risks. Simply because they are hard to replace, especially if they power important business processes in an organization. The risk of changing the status quo may be too high if the company takes into account the possibility of key data getting lost or corrupted.