International Journal of Management, IT & Engineering
Vol. 7 Issue 2, February 2017,
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A

GAURAV'S THEORIES AND LAWS ON PROGRAMMING AND APPLICATION DEVELOPMENT

Gaurav Kumar Roy*

Abstract

This research paper is mainly written to make few natural – digital laws and happenings clear to the software engineers, which they unknowingly using but still unknown of the fact that even computer world also holds some theories and laws which are universally correct in all fields of programming and software development or software engineering. I further requested the USA's Institute of Electrical and Electronics Engineering organization to take these laws for granted as these may prove useful universally in all cases of software engineering. Moreover, these laws are shaped and thought by me based on the previous 6-7 years of experience in this field and software engineering and programming methodologies and coding related book writers are welcome to add these laws to their chapters to aware every one of the traditional and never-changing fact that these theories and laws will guide and mould a coder or programmer well during upgrading or modifying any application.

Keywords:Laws of programming, coders, program, upgrade, complexitysoftware engineering application maintenance

* **Doctorate Program, Linguistics Program Studies, Udayana University Denpasar, Bali-Indonesia**

1. Introduction

Physics, biology, and medicine science have well - defined public explanations of various laws and theories put forward by many famous scientists. Even in simplified form, these provide guidance to the next generation in their research and scientific work. Furthermore, with the inauguration of such theories and laws, physicists and biologists are able to follow a specific path to approach in their respective field. Similarly, in the field of software engineering also, the coders and software developers need to take care and well aware of certain protocols which are universal nature every software or application by this way or that way follows in every field. Even these theories and laws can be applicable to web development also. There already exist some theoretical views which are SEMAT (Software Engineering Method and Theory) that drives a process to re-found software engineering based on a solid theory, proven principles and best practices. But the theories and laws I have put forward are in the nature of every software or application which programmers build with the change in time keeping in mind that the technology or implementation tool(s) may change but the theories and laws will be universally applicable with the growing ages.

2. About the Theories and Laws: -

Theories and laws can address any problem since theories and laws provide a platform based on some natural rules that explain the digital world with the real world scenario at some appropriate level of abstraction, but cheaply and without causing any harm. Theories and laws of computer science can provide answers to many questions that otherwise might be prohibitively expensive or impossible to portray. A general theory or law in software engineering field would ideally advise the development or coding, against costly error before the trial begins.

3. SOFTWARE ENGINEERING AND ITS BRANCHES: -

3.1 What is Software Engineering?

Software engineering is ultimately about psychology, how humans manage complexities of software. So software engineering principles are far more familiar to education and management theories than physical principles. Some software engineering background has solid math behind them: O(n log n) sorts are faster than O(n^2) sorts, or time complexity related concepts, etc. But mostly software engineering is about how humans think about software. How to organize things

so that maintainers can anticipate what is likely to change and what is not, preventing and detecting human errors, etc. It's a branch of psychology, sociology, and methodology.

3.2 What is Application Development?

The phrase Application Development can be defined as the activity of computer programming that involves the process of writing / coding and maintaining the source code and with a broader sense, the term can be depicted as the art that includes all that is involved between the conceptions of the desired software application through initial to the final manifestation of that application. Hence, you can say that application development may include new development methodologies, modification of source code, reuse the program or the elements within it, re-engineering, maintenance by updating or any other activities that outcome in the semi - finished or finished application. So, it is advantageous in the sense that application development provides an organized flourish in delivering products in faster, better and cheaper ways. There have been many studies and suggestion in improving the development process.

Application development projects are sometimes notorious for frequently changing and upgrading initial planning and specifications based on requirements of users. Also, after delivering the software product, updating and patching the software is a never - changing de-facto in software engineering, i.e. for better software development process, a software must have to be maintain by updating and / or patching that software. In order to identify the main reasons for changing specification during the development stage of a software product debates were started on LinkedIn project management groups.

3.3 Application Maintenance: -

Application maintenance in software engineering is the up-gradation or modification of an application after delivering the product, mainly for correcting faults and for improving performance and other attributes (like bringing down the patches). In other words, Software maintenance in software engineering is the alteration of an application product after delivery to correct faults, loopholes in order to perk up performance; a common perception of maintenance is that it merely involves fixing defects or enhancing features.

3.4 Product Marketing: -

It is the overall process required in transmitting a good or service to customers by proper marketing of product. Product marketing comprises of defining the scope of the product line, identifying potential markets for a better product, determining the best & most favorable pricing for the product, encouraging and attracting potential customers to purchase the product finally finding and opting the best distribution techniques and methodologies to deliver the product to customers or to sales site.

You might be thinking that why am I discussing about business management topics and expressing marketing strategy. The thing is my 4th law on Programming & Application Development is based on the attraction and natural selection of product which can sustain in the market for long time.

3.5 Product Marketing Strategies: -

An effective marketing strategy will help software vendors to define the overall direction and goals for marketing the application. The strategy should eloquent how you are going to deliver your products or services in ways that will and should satisfy your customers' need and requirements. Once you have defined and structure your customers or target market (software market), you need to start developing and implementing tactics or ways to reach them. The marketing mix will make up the tactical elements you will use to carry out your strategy and reach your target market.

The prime focus in making strategy of product marketing will be -

- Quality of your product or service
- The pricing of your product or service
- The promotion of your product of service
- Dealing with Product consumers
- Delivering of proper updates and patches

3.6 Software Anthropology: -

It is a termed coined by the writer himself, which depicts the behavior and study of various aspects of humans within past and present experience with any software and how the taste of users varies with time when using any specific software or what are the future requirements that the user is going to need, and why this variation took place and how can things be modified from the past experience both look-wise or design-wise and performance-wise or facility-based.

4. Theories and Laws proposed by Gaurav : -

My theories and laws on coding and application development ranges it's universal nature and meaning from application coding and development phase to its upgrading and deployment phase either in case of marketing the software product or after delivering to an organization or individual.

The writer of this research thesis proposed five (5) Universal Laws of Application Development and its programming which are common principles an application follows and as the nature of any application prevails as it continues its legacy. The Theories and Laws are described as follows –

4.1 1st Law on Programming and Application: -

It states that for every upgrading in a source code (for refinement and making its functionality better), there is an equal i.e. positive and opposite i.e. negative consequences. Positive effect can be derived in the sense that with the gradual upgrading of any source code within an application, the capability and working flexibility / choices and advantages may / will increase, with the decrease in simplicity of code i.e. the increase in complexity (both time and space) until and unless a proper measures or set of measures are taken to reduce complexity. Negative effect can be derived in the sense that with the upgrading of any source code, its source code may have increase in time and space complexity which can eventually give rise to slow compilation or slow performance and can even lead to increase / upgrade in the hardware infrastructure of the system, which may not be possible in all cases. There is a say that "Every good thing comes with something bad". So application developers can remove or eliminate those complexities and other disadvantages by taking proper measures.

In other words, a short derivation of this law can be like this – Programming and Application Development's 1st law states that for every up-gradation of any source code of any application, there is an equal and opposite consequences; which a developer should keep in mind.

4.2 2nd Law on Application Development: -

It states that the code of any program as and when gets updated, the later source code (after modifying the code) should to be familiar with the previous source code (before modifying the code) where there can be slight or heavy refinement in source code and addition of features, along with the similarity in logic with respect to previous one. The relationship & familiarity should be maintained as per the requirement of the user and this proves to be a universal and natural fact in application development world at the time any source code gets upgraded.

In other words, a short derivation of this law can be like this – the Programming and Application Development's 2nd law states that with the up-gradation of a source code, there remains a perpetual relation as well as familiarity between the old or previous source codes with the upgraded or modified source code.

4.3 3rd Law on Programming & Application Development: -

It says that, behind every creation and / or up-gradation of application, the application has a correlation in parallel with the hardware requirements as well as existence of hardware architecture and the platform on which it will run. So, basically the programmer should think, about whether the developed product will run on that particular architecture for which it is intended and must think beyond about the user who is going to use it is comfortable with the intended product (i.e. user can able to handle with ease or not) every time before developing and upgrading the code of an application.

There are two options here, either you build a product in such a manner that it runs on every platform irrespective of the hardware and platform architecture (like Java, C++, Linux OS, Atom etc) or the application developer have to keep in mind about the existing platform on which the application will run.

So, this becomes a natural phenomenon of every application that an application will always be made to work on a particular system (IBM-370 or Macintosh) or hardware or OS or platform and thing may or may not initially come to the developer's mind but either intentionally or unintentionally this becomes a major and universal fact behind every application.

In other words, a short derivation of this law can be like this – the Programming and Application Development's 3rd law states that behind the development or up-gradation of an application, the application must have a correlation with the hardware or the platform on which it will run (perform its tasks) either intentionally or unintentionally.

4.4 4th Law on Application Development and product marketing: -

Though the theory of "Natural Selection" was proposed by Charles Darwin in his theory of evolution which was first formulated in Darwin's book "On the Origin of Species" in 1859, where it states that the process by which organisms change over time as a result of changes in heritable physical or behavioral traits. Changes that allow an organism to better adapt to its environment will help it survive and have more offspring, i.e. "Survival of the Fittest". Natural selection acts in preserving and accumulating minor advantageous genetic mutations. Suppose a member of a species developed a functional advantage (it grew wings and learned to fly). Its offspring would inherit that advantage and pass it on to their offspring. The inferior (disadvantaged) members of the same species would gradually die out, leaving only the superior (advantaged) members of the species.

Evolution by natural selection is one of the best substantiated theories produced, supported by evidence from a wide variety of scientific disciplines, including paleontology, geology, genetics and developmental biology. I have taken support of Sir Darwin's proposed theory and come out with the conclusion that even software applications also follow this law to sustain in the software market.

The natural selection also becomes applicable in case of software development and marketing when it comes to a note that the software which can adopt to the environment and possible requirements of the market can continue its legacy (like: Windows OS, Linux, Microsoft Office,

Adobe Products etc) and the ones which are unable to satisfy consumer needs and requirements in the long run will eventually come to a termination. This is also an universal truth of the software market, where the quality of the product, optimization in pricing, time-to-time deliberation of updates and patches are the primary focus which are kept in mind while selecting a product.

The statement "Natural selection is the preservation of a functional advantage that enables a species to compete better in the wild" can be applied in case of software product marketing also. According to the 4th law, natural selection goes on preserving the best application that enables the user to facilitate better in all respect (price-wise, quality-wise, performance-wise, multi-requirement wise) and compete in the fast growing viable software market.

4.5 5th Law of Application Security: -

It states that for developing every good application, there remains some security postures, which need time-to-time updating and patching when any bug encounters.

Moreover, software engineering must incorporate the fact that when an application is having an admin-user scenario or multi-admin with multi-user application structure, then there must need to have some proper layers of security within that application, which can make secure connection and increases the liability of application for better marketing facility.

Though today's most of the application is having security measures already taken, but security is obscurity, so it need timely checkup and with the upgrading of architecture, the security measures should have to be properly upgrading.

5. Conclusion

These laws may be followed by application developers or programmers but unknowingly. So, now applying these laws, application developers can construct or develop their application with the thing in mind that these universal laws with or without developers' conscious will run in this nature of every application and man make things better if the developer kept these in minds while developing any application.

References:

[1] Victor R. Basili. The experimental paradigm in software engineering. In Experimental Software Engineering Issues: Critical Assessment and Future Directives. Proc of DagstuhlWorkshop, H. Dieter Rombach, Victor R. Basili, and Richard Selby (eds), published as Lecture Notes in Computer Science #706, Springer-Verlag 1993.

[2] Mary Shaw. Prospects for an engineering discipline of software. IEEE Software, November 1990, pp. 15-24.

[3] Wohlin C. An analysis of the most cited articles in software engineering journals-2000. Information and Software Technology, 2007

[4] Buse R P L, Sadowski C, Weimer W. Benefits and barriers of user evaluation in software engineering research. ACM SIGPLAN Notices, 2011