**Dinesh Gonuguntla**

**0918700**

**Final Project**

**No Silver Bullet Essence and Accidents of Software Engineering**

The No Silver Bullet Article Essence and Accidents of Software Engineering by Frederick P. Brooks discuss that we don’t see a No silver bullet recently because there is no single development happened in either technology or in management technique. In this article he will try to show why, by examining both the nature of the software problems and properties of the bullets proposed. The progress would be made stepwise, at great effort, and that persistent, unremitting care would have to be paid to a discipline of cleanliness.

The essence of a software entity is a construct of interlocking concepts: data sets, relationships among data items, algorithms, and invocations of functions. He believes that the hard part of building software to be specification, design, and testing. We still make some syntax errors, to be sure, but they are fuzz compared with the conceptual errors in many systems. If this is true, building software is always hard. There is inherently no silver bullet.

The most powerful stroke for software productivity, simplicity and reliability have been the progressive use of high-level languages for programming. The accomplishment of the high-level language is free a program from much of its accidental complexity and it eliminates the whole level of complexity that was never inherent in the program at all. This language can do is to furnish all the constructs that the programmer imagines in the abstract program.

One must be very careful when defining an Abstract data type and hierarchical types. The concept of Abstract data type is that an object's type should be defined by a name, a set of proper values and a set of proper operations rather than by its storage structure which should be hidden. Hierarchical types allow one to define general interfaces that can be further refined by providing the subordinate type. Both are orthogonal one may have hierarchies without hiding and hiding without hierarchies. Both concepts represent real advances in the art of building software. For both types, the result is to remove a higher order kind of difficulty and allow a higher order to express of design.

Each software organization must determine and proclaim that great designers are as important to its success as great managers are, and they can be expected to be similarly nurtured and rewarded. There are some tips to increase the great designers. As early as possible, systematically identify best designers. The best is frequently not the most experienced. Assign a career mentor to be responsible for the development of the prospect and keep a career file carefully. Give them the opportunities to interact with and stimulate each other.

Automatic Programming always been a euphemism for programming with a higher-level language than was presently with an available to a programmer. He argues that it’s a solution method not the problem, whose specification must be given. The favorable properties are: The problems are readily characterized by relatively less parameters. There are many known methods of solution to provide a library of alternatives. Extensive analysis has led to explicit rules for selecting solutions techniques, given problems parameters.

Graphical Programming is a favorite subject for PhD dissertations in software engineering is graphical or visual, the application of computer graphics to software design. These approaches are postulated by analogy with VLSI chip design, there computer graphics plays so fruitful a role. The flowchart is a very poor abstraction of software structure. But it is best viewed as Burks, non-Neumann, and Golds tine’s attempt to give a desperately needed high-level control language for their proposed computer.

**Primavera Gets Agile**

The Article Primavera Gets Agile: A Successful Transition to Agile Development by Bob Schatz and Ibrahim Abdelhafi discuss that Primavera systems will have the ability that helps the customers and manage their projects. Their development organization will employ 90 Business Analysts, Programmers, Testers and Project Managers. Agile Software Development is the best solution to build the software and increase the quality of life for everyone on the team.

Agile methodology is a process of continuous learning and an improvement. To learn this one can, need an intense focus, hard work and discipline. Working as a team they can come across many things like their process and products. Every successful product they delivered revealed their understanding of working as a team. To improve their software in higher success rate Software development community needs to adopt an Agile.

People think that developing the software for a company is too easy. But like everyone, they will work hard to deliver a defect less software. They have roots in Construction and engineering. They used a Waterfall development model approach to release a software. In Agile they need to follow a cycle. To finish projects on time they will work on late nights and weekends. In Primavera Project managers used a command and control philosophy, they will take decisions on their own without contacting developers doing the actual work in the project. Because of that, relations between Project managers and Developers will not that good. After changing the Leadership in 2001 he looked an overview of many issues within the company. The main point he mentioned is don’t depend too much on higher positions and departments need to resolve their problems.

After changing the leadership, they created a new development team which can provide services to their customers more effectively. For sharing values, they created a Fish. Software development professionals are doing the same work again and again. Extra hours to design a software and working on week Ends, but they don’t really get an appreciation for their work. In March 2003 they released a software. Development team taught the release went well, but others taught the release was not good lacked quality issues and some features. After they got a feedback they increased their developers and they taught they needed to implement an Agile.

To overcome these, they found out a solution I.e. SCRUM which has Scrum teams, Sprints (30-day iterations) and 40-hour work weeks. In the beginning, Each Scrum Team and Product Owner will work together to set goals in each sprint. Basically, the Product owner will be from the marketing department. He will tell the features what all they need to implement to satisfy the end user’s needs.

Each Scrum team has Scrum Master, Business Analyst, Developers and Testers. Scrum masters will set up a meeting every day. Initial minutes everyone needs to discuss what they have done yesterday and what they are going to do tomorrow. If they are facing any obstacles during their work Scrum masters will need to resolve those problems. It depends on the Scrum master how many days the Sprint would be. At least the sprint has 2 weeks. At the end of every Sprint, they have a retrospective call, to discuss their all pros and cons what they faced during the sprint.

They don’t want to tell people that they are implementing a SCRUM in our projects. They don’t want to make them nervous. Instead, Ken proposed to do a workshop for all the Dev teams and then we do a training session to certify some SCRUM masters. After presenting that, scrum reactions were mixed some people adjusted and some people confused what are all our roles and responsibilities in the new team. Regardless of whatever it is they were committed to implementing the scrum.

Some of the Tips for moving to Agile.

First thing is they need to find out a sponsor who will going to implement Scrum in their projects and they need to move agile process.

Second, this is the need to take a coaching from the outside who are expert in this field.

Third things are Teamwork is more important when we move to agile. Project managers need to shift their focus from tasks and assignments to team dynamics. With the help of teamwork, we can grow better in an agile process.

For understanding, purpose don’t change the agile process to the old way. People may slip to their old way of doing things.

Agile is a bottom-up fashion for adopting that we require an executive support.

Without working Overnights and Weekends we can able to deliver high-quality software.

There is no silver bullet in Agile. Agile is not an easy way to adopt. The best way is to learn and improve upon our experiences.

There is a tool for the scrum to track all the data. It will show them how many hours they have spent in each sprint. Each Scrum team divide their tasks according to their requirement. Each scrum must update their work at the end of the day. If the story is in work means they will update in progress, if its completed dev they will mark as completed by updating how many hours they have spent. At the end of each sprint, they will revive their product backlog and move to in progress according to the higher priority stories in the backlog. And then the scrum team start their next planning session to discuss the new user stories.

By using the scrum, we can able to measure our work very easily. After every release, we get to know how the release went well and how the end users are using the application. After using the scrum our quality was increased to 30 percent. We can able to move beyond in work (by starting the next sprint User stories in advance) by using the scrum.

After adopting the scrum, in their entire development cycle, they never worked overnight or weekends instead developers have time to help their scrum teams to achieve the sprint goals. Which make them team members was more enjoyable and began to trust each other. They will get a chance to work closely with product owners and stakeholders which help them to get influence on the product.

Each sprint review they make their user stories slices that makes product owner easier to review and focus on higher vales items. In most of the cases, 50 percent of this story implemented was enough to meet the business needs, as a result, the product owner drops the remaining thing or moves it into the backlog. Scrum gives them more control to their stakeholders of the release backlog because it will let them provide more inputs in the early development process.

There aren’t many rules in the scrum, but you need to follow the one that exists. They set clear criteria what constitutes a full feature, and only the features that fit the criteria are shown at the sprint review to stakeholders. They gave priority to the codebase’s maintainability and extensibility by having the product owners add technical maintained items to our current release backlog. This method makes sure that infrastructure items don’t lose and addressed on every sprint as part of their overall release.

In the end, they are adopting some extreme programming practices such as test-driven development and pair programming. After implementing Test Driven Development their defects have dropped to less than 10 per team, which represents an overall 75 percent improvement to their previous releases. Because of these team can be confident.

The lessons they have learnt from the scrum is as a team, building software is a continuous learning process. We must have the discipline what we are doing and don’t be afraid of making any changes. By doing all this they can able to change their process step by step and continued to make improvements every single day.

**Scaling Agile**

The article “Scaling Agile” by Christof Ebert and Maria Paasivaara discusses that agile software development has become conventional. Industry scaling agility for distributed teams, large projects, require scaling agile practices, which agile scaling framework provide. Agile practices have evolved steadily, in the early 1990’s Microsoft invented most of agile.

The key milestone was internet explorer, which Microsoft redeveloped entirely in the 1990’s to allow for flexible, scalable evolution. The starting agile manifesto was experienced based on Microsoft, IBM and others they collected primarily principles and practices. Scaling isn’t easy. Large projected are distributed across globally and many teams that need to collaborate and coordinate.

In early years agility was more often a dogma than a logical yet flexible framework. The V development model is expanding to a W shaped model of incremental development and continuous delivery. W model starts with agile planning (involves frequency of deliveries, increments, and train of release). Then it proceeds to incremental design, verification, and integration. Services based companies allow adjusting to services and micro services and evolving towards DevOps practices. E.g. (over the air upgrades of modern automotive on-board infrastructures). Current approaches for scaling Agile.

Business model -from the static building and selling to dynamic composing and continuous delivery. Governance- Simple Information technology systems and disposable apps to interwoven quality assurance with auditable product liability, functional safety and cybersecurity. Process- from V model with heavy release cycles to scalable agile process. Development- from components and functions to services. Architecture- from localized features to service-oriented patterns.

In a recent survey, almost 30 percent of the respondents said that they used the scaled agile framework to provide scaled agility. Many of the practitioners think the scaled agile framework is too heavy and complex. CMMI tries to include all best practices but it doesn’t provide guidance on how to scale down. Large Scale Scrum and Lean Scalable Agility for Engineering have addressed this high complexity.

An agile framework is implemented in Napa and Comptel. Napa was adopting the agile framework was slow and gradual whereas at Comptel both business lines planned and transformed the organization in a couple of months. Both companies customized the framework to suit their organizations. Agile framework determines the specific roles such as system team, product manager, system architect, release management team and deployment team. The goals both companies reached is More frequent and predictable release with better quality, Improved visibility and communication across the worldwide distributed organization, better synchronization and collaboration across the teams and better alignment across the organization.

The increased demands for high quality, quick delivery, reduced costs, and flexibility pushed agility to practically all industries. The user wants the same feel and comfort when they are using mobile devices. Agility has come in real-world development beyond more software applications. Agile service delivery models combining DevOps, micro-services, will allow functional changes far behind the traditional V model.

Introducing the agile in the development means changing the culture and mindset of the people in that organization and it requires a long-term commitment, big investments and customization. There are no barriers exist for using agile technology. Some of the projects showed that agile technologies are adaptable and adaptable to large projects, safety-critical systems, globally distributed teams, and hardware and system engineering. Scaled agile doesn’t about tailoring a model, it’s about ensuring that the agile culture reaches the engineers and their management.

**Extreme Programming from a CMM Perspective**

The Article Extreme Programming from a CMM Perspective by Mark C. Paulk discuss that Extreme programming is an agile methodology that some people support for high speed, volatile world of internet and web software development. Extreme programming is a disciplined process compared to Software capability maturity model. XP has better engineering practices that can work well with the CMM and other structured methods.

The SW-CMM was developed by Carnegie Mellon University Software Engineering Institute as a model for building the organizational capability, Presently, it has been adopted in the Software Community. SW-CMM informative materials focus primarily on large projects and large organizations. XP target is small to medium-sized teams building software with rapidly changing requirements. The teams of XM have typically fewer than 10 members. XP team typically deal with requirements changes through an iterative lifecycle with a short cycle. This XP life cycle has four basic activities- coding, testing, listening, and designing.

The XP practices have 12 basic elements: --

Planning Game: Fastly determine the next release’s scope, combining business priorities and technical estimates. The customer decides scope, priority, and dates from a business side, whereas technical people estimate and track the progress of these.

Small Releases: Keep a simple system into production quickly. Release new versions on a two-week cycle.

Metaphor: Guide all development with a simple, shared story of how the overall system works.

Simple Design: At any given moment design as simple as possible.

Testing: Developers continually write unit tests that must run flawlessly whereas customers write a test to demonstrate that functions are finished. “Test, the code” means a failed test case is an entry criterion for writing code.

Refactoring: To remove duplication, improve communication, and simplify we need to restructure the system without changing its behavior

Pair Programming: All prod code is written by two programmers at one machine.

Collective Ownership: Anywhere at any time anyone can improve any system code.

Continuous Integration: Integrate and build the system many times a day. Continuous regression testing prevents functionality regressions when requirements change.

40-hour weeks: Never overwork two weeks in a row and work no more than 40 hours per week whenever possible.

On-Site Customer: Put an actual user on the team full time who can answer all the questions.

Coding Standards: Have rules that emphasize communication throughout the code.

Extreme Programming encourages collective ownership anyone can change any piece of code in the system at any time. The XP mainly emphasis on continuous integration, continuous regression testing, and pair programming protects against a potential loss of configuration control. For good software engineering thinking about testing early in the life cycle is a standard practice, though it’s too rarely practiced.

The SW-CMM both are focusing on the management issues involved in implementing effective and efficient process and on systemic process involvement. The both can able to create synergy, particularly in co-existence with other good engineering and management practices. Mainly XP focuses on technical work, whereas the CMM focuses on management issues.

Most of the extreme programming consists of good practices that all organizations should think of it. Although its concepts have been here for decades, adopting simultaneous engineering practices changes your product building paradigm. XP gives a system perceptive on programming, just as the SW-CMM gives a system perceptive on organizational process improvement. Those whose organizations want to improve their capability they should take advantage of the good ideas from both and do some do some assignment in selecting and implementing those ideas.

**Processes for Producing Secure Software**

The Article “Processes for Producing Secure Software” by Noopur Davis and Watts Humphrey summarizes the work initiated at the National CyberSecurity Summit. This Summit representing Industry, Academia and the Department of Homeland Security. Basically, this summit is to identify the key problems and recommendations identified by the Security across the Software Development Lifecycle.

Total Vulnerabilities security incidents reported to the Computer Emergency Readiness Team Coordination Center rose from 171 in 1995 to double in 2000 whereas in 2003 its saw a major increase in the software vulnerabilities reported. For improving the software engineering practices and the process can guide the design of secure software. Whereas process improvement produces software releases with very few defects, affordable development costs, lower maintenance cost and intensify product reputation. Whether we purchased the software or built in the company all parts of the security systems are relevant and the specifications must define the functionality and be free of vulnerabilities. Another necessity for secure software production is correct design and implementation.

The reasons for producing secure software is first by doing these in all levels like team, individual, project, organization and on-site can complicate matters. Second, most people share the perception that functionality is more important than quality or security. Lastly, many organizations believe that developing high quality and secure software is too expensive. Microsoft has found that half of their software security problems are due to design flaws. For avoiding that design flaws they require high levels of security and design experts. The significant changes needed to produce secure software is not easy for many organizations.

The Team Software Process(TSP) is an operational process for use by software development teams. As of date, many organizations are using the TSP and a recent study of 20 projects in 13 organizations showed that teams using it produce software with an average of 0.06 delivered design and implementation defects per thousand lines of new and changed code produced. These teams delivered their products an average of six percent later than planned, which shows that many software projects are late or canceled. A team at a major software vendor has used the Team Software Process the results were good and they produce the defect-free software in which they didn’t find any security defects during the audit time or after months of several uses.

For software production Formal methods are mathematically based approaches, they use the mathematical models and formal logics to support strict software specifications, design, coding, and verification. Correctness by construction incorporates the formal methods which do the early verification and defect removal throughout the software life cycle. The Correctness by construction method produced the defect-free software in some of the projects completed between 1992 and 2003.

For designing the Secure software principles alone are not sufficient but they can help guide development practices. Some of the eight secure software development practices which were proposed in 1974 by Jerome Saltzer and Michael Schroeder's are: Economy of mechanism- When implementing the design should be as simple as possible. So that everyone will understand easily. Fail-safe Defaults- Access decisions should be based on permission rather than exclusion. Complete Mediation- Every access to every object must be checked for authority.

Open Design-There shouldn’t be secret in any designs which they are designing. Separation of Privilege- Wherever its possible, for a protection mechanism that requires two keys is more strong and flexible rather than one that gives access via a single key. Least Privilege- Each program and user of the system should operate using the least set of privileges necessary to finish the job. Least Common Mechanism-The amount of mechanism common to more than one user and depended on by all the users should be minimized. Psychological Acceptability- The interface for end users should be ease of use so that users automatically apply the projection mechanism correctly.

In order to create secure software developers must follow these basic guidelines and checklists. Validate input and output. Ensure secure failures. Keep it simple. Use and reuse the trusted components. Provide only absolutely necessary privileges. Allow encryption of all communication. Never share the passwords in plaintext. Ensure that software is delivered and installed securely. These guidelines must be considered during the design, implementation, testing, delivery, and installation. Languages such as C# and Java are good for developing securing software although we have more choices present.

The Department of homeland security has welcomed the National cybersecurity partnership recommendations and considering which initiatives it might support. Some of the recommendations are accomplished and pursued. The organizations that use processes that produce software with very fewer defects, in-depth security expertise, practices and that focus on continuous improvement will be hugely successful in producing more secure software.

**Secure Software Development by Example**

The Article “Secure Software Development by Example” by Axelle Apvrille and Makan Pourzandi discusses the security concerns at each step of a project’s life cycles. In CERT coordination center database and Bugtraq, mailing lists gives an security alerts every day, that gives an concern of security holes, even for nonsecurity-related applications like Apache. This concerns gives an extra pressure to software developers especially on how to build secure software. The aim is to introduce developers to several security tools and give a complete process for secure software development. Certainly, we cannot give assurance a security bug-free program, but this case studies will shows the types of situations that developers face.

Methodology: Basically projects life cycles follows an iterative process of analysis, design, implementation, testing, and maintenance. In this article, we will mainly focus on how to adapt this methodology to produce an trustworthy application. We insert security concerns in each step of the life cycle expect the maintenance stage is omitted.

Analyzing Security consists of several steps. First, we must define the projects security environment and objectives, then we can list the application potential threats and prioritize them. A security environment describes the context in which the software is expected to evolve. PICO is a client-server application based on exchanging messages over the internet by group of people. In this environment, the presence of the information about users and messages is exposed to malicious software. This type of threats of a given security environment is exposed lead us to security objectives, which are high-level security issues. Integrity protection of exchanged messages and client authentication stops the attackers to modifications in PICO.

Developers should implement the threat model so that it can identify possible threats in a given security environment. The most common way of finding threats is to use the stride categories for spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege or 49 attacker patterns identified elsewhere. For security policy, there are some security requirements that we can prioritize them according to the information security.

PICO has implemented the following security requirements for higher to lower. To use the services users must be authenticated. Give an assurance that messages and presence are encrypted, expect the user no one else can see their messages and their presence. The sender of the message should be properly authenticated. The confidentiality of the message and subscriber database privacy should be guaranteed. Some of the best know formula for risk evaluation is Risk=Critically \*Likelihood of occurrence.

Designing Security: -Most of the design methodologies must be adapted to security because we need to be able to express security concerns such as critical data, communication security. We design and model our software security aspects with UMLsec (it’s a UML profile that extends the UML standard by adding stereotypes, tagged values, and constraints).

Testing Security is quite difficult because it doesn’t follow the standard rules that apply to software validation. For integrity and confidentiality previously, developers implement their own secure communication from scratch, they promote the reuse of existing solutions, but these solutions cannot be integrated blindly without tests.

The development of secure application is a case for compliance. They taught their 100 lines of c code would prove to secure immediately, but they spotted several vulnerabilities right way and some are not yet seen. To implement the design secure systems there are some solutions exist, but it’s not hundred percent accurate but it will give some results. Code review is the best one for secure testing rather than the tool testing, although code review will take more manpower and time consuming and the reviewers of the code review depends on the professionalism and degree of competence.

**What Can Agile Methods Bring to High-Integrity Software Development**

The “What Can Agile Methods Bring to High-Integrity Software Development” by Roderick Chapman, Neil White, and Jim Woodcock discusses that for software development there is much interest in Agile Engineering. Agile promotes its flexibility, leanness, and ability to manage changing requirements and express contempt the waterfall approach. Attackers criticize Agile’s free for all.

In Altran company in the United Kingdom, they used disciplined and planned engineering, especially when it comes to high-integrity systems that involve safety, and security. Agile is an anathema to high-intensity systems development. In my view this is correct Because I worked on Agile it’s so flexible and I can’t able to imagine without this in my work and I am addicted to it. Like whenever any topics that come related to Agile I will take that first irrespective of other things which are there. As their projects develop they absorb change and respond to defects like any other. This put to good effect in the Multos CA project, which they have planned for seven iterations but after 13 iterations they have delivered owing to a barrage of changing requests. They find out that change and iteration were the heart of agile manifesto.

There are so many people who think that Agile as beginning with XP, but its roots are much older. Most of the core practices for extreme programming were established a long time ago and their combination and rigorous practice was novel. Use of Agile has been reported by Thales Avionics, while SINTEF has reported success with safe scrum. For each issue in agile, they start with a brief recap of the practice in question, then go on to describe the issues followed by their ideas and experiences in overcoming it.

Meyer calls some of the practices in agile were brilliant in his summary analysis (Its calls for continuous integration with regression test suit and a test first development style with every function is associated with specific tests) But agile assumes that dynamic test is the principal verification activity. In high integrity development, they use different forms of verifications, including checklist driven reviews, automated static verification, traceability analysis, and structural coverage analysis.

Using the SPARK toolset developers precede commit, build and testing activities with static analysis. When they are working on an isolated change developer can reproduce the proof of the entire system in about 15 minutes on their desktop machines to change a single module. Agile projects have a “don’t break the rules” mantra on iFacts its “don’t break the proof.

Their principle in meeting nonfunctional requirements is system architecture including redundancy and separation of critical from noncritical. Agile requires a single active sprint delivered immediately to the customer. Usually, every sprint has at least 2 weeks and at the end of the week, they are delivering their project to live or production. Build N: In operation with the customer, used to report bugs. Build N+1: The current development sprint.

iFacts uses a deeper pipeline and multiple iteration rates, with at least four builds in the pipeline. Build N, Build N+1: Undergoing customer acceptance. This process to subject to regulatory requirements so it can take months. Build N+2: In development and test. Build N+3: undergoing requirements and formal specification.

No clash: Continuous integration, verification driven development cycle, continuous integrating testing, and explicitly planned iterative approach. Agile Assumptions: Customer decision making power and tempo, availability of plentiful test hardware and commercial and contractual model needed to procure agile. Opportunities: We need to adopt formal languages, automated synthesis, and static verification as part of the deployment pipeline.

**Lying on Software Projects**

The article “Lying on Software Projects” by Robert L. Glass, Johann Rost and Matthias S. Matook discusses that lying isn’t a new phenomenon. In software profession especially in Project estimation and Status reporting departments more often encountered a lying. Whenever lying happens developers often know about it even when the management doesn’t.

In Estimation the most predominant form of lying on software projects was estimation. 66 percent of respondents said that they had experienced lying. The typical estimation lie was the largest number, 18 percent, said the lie was off 50 percent. 18 percent said that it had happened on 50 percent of the projects they are familiar with, whereas ten percent saw it happening on the project. The answers from the respondents on lying of the projects are. 21 percent said that we need to Improve management techniques, such as better time tracking and control. 18 percent were change who does the estimate, such as involving the developers. 15 percent said improve the estimation process, such as using independent estimators and ten percent said improve communication, such as not shooting the messenger.

The second most predominant form of lying is Status Reporting were 65 respondents said they experienced lying. Fifty percent noted overly optimistic reporting, whereas only two percent noted overly pessimistic reporting. In Status report Twenty-one percent said that lying happens on fifty percent of projects. Regarding motivation, 66 percent said that they lied to tell management what they want to hear: 19 percent said it happened on hundred percent of the projects and 15 percent said it happened on 50 percent. 44 percent said that it was to hide the bad work:21 percent said this happened on 20 percent of projects and 16 percent said it happened on 50 percent of the projects.

The third most form of lying was political maneuvering. 50 percent had encountered this form of lying:24 percent said that it happened in 10 percent of projects (this has the lowest figure for any reasons they had identified). Twenty-one percent said the lies where told Project manager, 13 percent said to the lead, 10 percent said to the customer, 7 percent said to the developer and marketing.

The last one and the rarest form of lying is hype. 57 percent had encountered this: 16 percent they saw it on 30 percent of projects. To whom the hype was told Eighteen percent said management and the lead. 15 percent the customer, 13 percent the developer, and 10 percent marketing.

They received a mixed feeling about whether lying was morally wrong, with most of the experts agreeing that certain kinds of lies aren’t but that others are. In every case, developers on a project knew more often than anyone else that a lie was being told. Estimation and Political Maneuvering lies came more often from the management, for status report its form Leads and hype came out from marketing.

**A Fresh Perspective on Old Problems**

The article “A Fresh Perspective on Old Problems” by Ryan Fleming shared the experiences that, he had during the one-year internship as a programmer in the information service department. Initially, he spent the time in debugging, extending and updating the company in house billing system. Later he had an opportunity from programming practices to team organizations. The task which were given to him are solo missions and he soon began helping with other developers to debug their program.

During one debugging session, the developer said I don’t know why the program doesn’t work. He suggested looking at the manual or asking other developers, but he is doing guess and check code changes to work his code. Eventually, the code worked the developer went happily on his way and he went through the manuals or an explanation. Neither of us don’t understood what was wrong how he fixed it.

He encountered the same process with a different developer. After few iterations, we arrived a working solution. Ryan want to find out why it worked but the developer wants to move on. When Ryan protested the developer replied, hey programming isn’t science It’s more like magic or alchemy. Whether programming is alchemy, craft, science software developers debated for at least a decade. I believe programming is far too comprehensible and repeatable to be considered alchemy, most developers would agree this is a craft, but fewer will go far to say its engineering. Sometimes programmers run into mysterious and seemingly unexplainable, but it’s the failings the individual programmers that makes programming seems mysterious. Every time they fail to understand the solution we undermine, programming as a respectable craft.

Developers pull towards the tools and tricks rather than the process. By trying to produce a product faster with the help of tools and tricks a guiding process can decrease productivity and lead to chaos. Applying an effective process to technical projects seems difficult and elusive. Software developers are frequently involved in poorly run projects. Any development process is primarily an interpersonal undertaking, not a technical one. After spending money on new tools, the employees realize that they still suffer the same development issues. Buying more and better tools requires a capital investment that yields little change in productivity.

Ryan got a chance to work in the IS department and he decided to feed the IS director any information that he could find about department organization. Most IS programming tasks are short with very restrictive deadlines. Applying a real software engineering process to an IS department would require so much documentation, design, and development overhead. They used the automation in the IS department to automation all the requests which they receive.

Customers know what they want from the developers, but they have no idea how to communicate their desires to help developers to deliver a solution to them. Developers must extract the technical content from non-technical communication so that they can create some specification on which they can able to build the product. This communication makes the identification of product specification so difficult.

Many software professionals believe that pursuit of quality is the key of higher productivity, but some try to improve productivity by shortening the QA activities. QA activities improves the work performed by the technical staff. This reduces the need for rework which means higher productivity.

Communication is important when you are talking about a technical stuff. If not, the complex ideas come from is confused. Be sure when talking about technical stuff everyone involved shares similar interpretations, otherwise avoid technical jargon. A picture is worth a thousand word is applying to technical environment, one good diagram can save much verbal and documentation energy. Actively listen and grasp the topics and clarify what you don’t understand. Don’t interrupt, it confuses the issue locks people out of the conversation and is simply rude. Coworkers appreciate effective communication more than many technical stuffs.

Always defend your ideas. Leave fighting for what right to philosophers, lobbyists, and upper management. Professional and technical issues are complex, what right is a matter of perspective. your reputation is based on your prospective. Software engineers always must learn new tools, so that they can able to secure job. Companies that lack current professionals can’t take the advantage of new technologies to increase productivity, reduce operating costs and move into new markets. Company has to offer to learn new methods and technologies form seminars, educational institutions, and in-house training.

**Software Engineering for the Internet of Things**

The article “Software Engineering for the Internet of Things” by Xabier Larrucea, Annie Combelles, John Favaro, and Kunal Taneja discuss that from Akin to Mania they were witnessing a software developer gold rush around the IoT. Neither research nor industry in immune for the fever. On the other hand, researchers have been concentrating on securing end-to-end communication. A survey reported that 31.4 percent of the surveyed organizations had launched IoT solutions. However, a DZone survey reported that 87 percent of the organizations were unconcerned or just a little bit concerned about the new IoT paradigm.

MIT Sloan Management Review conducted a survey on business executives, Project Managers, and IT professionals to understand the IoT challenges and opportunities. Customer Satisfaction: Managing networks of connected devices is influencing relationships between an organization, its customers, and suppliers in many ways.

Organizational Aspects: Many organizations need experts to take upper hand of IoT projects, including in data analytics and security. Variable Costs: The Variable costs in IT projects are very low whereas in IoT it is an actual network of real physical things that can grow unexpectedly. Social Implications: They don't have a clear picture how people will react to the IoT devices. Frank Gens predicted that two-thirds of IoT networks have a security breach.

There are no consolidated set of software engineering best practices for the IoT has emerged. The industry needs guidance to engineer the new generation of scalable, highly reactive, often resource constrained software systems characteristic of the IoT.

Developing IoT systems implies "new development processes," and that embedded developers must increasingly adopt agile and continuous practices. Past Software engineering techniques can be harnessed and adapted to the challenges of today's IoT.

The IoT as made of networked sensors and smart objects whose main purpose is to measure or control or operate on an environment in such a way to make it intelligent, usable, and programmable and capable of providing useful services to humans. NIST defines smart systems as co-engineered interacting networks of physical and computational components. Cyber physical systems have following technologies the IoT, the industrial Internet, smart cities, smart grids, and anything that has smart for example homes, buildings, hospitals, cars, applications etc.

The IoT will bring not only enormous opportunities but also enormous challenges. It's up to the software community to equip itself to face those challenges.