# Egypt–SPIN Newsletter

**Issue 5, Jan. – Mar., 2004**

Sponsored by SECC

## From the Editor

Welcome to our 5th issue of Egypt –SPIN newsletter. In each issue we are trying to put together relevant information in the form of articles and recaps from the previous 3 months events hoping to provide our members of Egypt – SPIN with information to support their current interests.

In this issue you will find different authors from the academia as-well as the industry sharing their knowledge and experience with the community:

Eng. Ahmed El Shikh is introducing the new hot **eXtreme Programming (XP)** methodology. The article explains the idea behind its success and discussing its suitability to Small and Medium Enterprises (SMEs). Ahmed explains briefly the methodology and concludes by the factors that should exist to make XP successful.

Dr. Hoda Hosney is suggesting a very interesting **framework** that establishes collaborative partnerships between industry and academia aiming to strengthen industrial and research capabilities and to foster the professional development of students and practioners. Electronic online training and online repositories is the proposed framework that fit with the speed of change in technologies.

Eng. Madiha is explaining what is meant by **software testing** in a phase oriented explanation. The article discusses the expected outcome form the testing process in each phase from the requirements till the maintenance and support.

Eng. Omar Kamal is writing about essential **software testing techniques** that should be a part of any software engineer's knowledge. The article is useful for both software engineers and project managers. From the point of view of the author, the techniques are useful even if it is not directly used.

Eng. Sameh Zeid shares with the community **ITSoft's experience in Software Process Improvement (SPI).** The article explains how ITSoft adopted the Software Engineering Institute (SEI) IDEAL® approach as a methodology for (SPI). Sameh concludes by listing a very useful short list of lessons learned from ITSoft journey of continuous improvement.

Eng. Nermine Faltas is sharing with us her experience in **improving the effectiveness of the software testing process**. She emphasizes on testing resource dedication, testing support tools, and early engagement of software testing in the software lifecycle. Nermine explains how those factors drive the effectiveness of the testing process.

We hope we succeed to give you an idea about what is going in our community. Please write to the editor your comments about our progress. We always ask you to submit short articles for publication that deal with your experience in defining, developing and managing software efforts as well as process improvement experience. Remember that our goal is to encourage an interchange between our readers. You can email spin@secc.org.eg or omkamal@yahoo.com

## Table of Contents

## It is still possible for SMEs to follow a formal methodology “eXtreme Programming, XP”

**By: Ahmed S. El-Shikh**

A frequent asked question that always appears in the SMEs is: “Is it possible to follow a formal software production methodology, even with the limited number of available people and resources?” Two different and even opposite answers can be heard for this question, the first will say “Yes, of course. It is possible and also a must to follow a complete generic methodology”, the second one says “No, it is impossible. We do not have the required resources or time to go through all these complicated practices”. If you are a senior manager, who are concerned to take the decision, be careful in both cases. The first answer belongs to the innovators that it is usually dangerous to follow them in all cases, and the second belongs to the laggards that they usually wake after the crises already happened. So you need to find your way in between these two alternatives.

A "Software methodology" can be defined as a set of rules and practices used to create computer programs. One of the well known methodologies is the “Enterprise Unified Methodology, EUP” which is the enhanced version of the “Rational Unified Process, RUP[1]”. The EUP is suitable for the elephant size companies which have enough number of people to perform the required roles and can provide the needed resources. Also, EUP and similar methodologies required a high level of maturity, which is usually not the case in SMEs, to own the ability to follow the well defined set of roles and detailed practices, i.e. to follow a complete software engineering roadmap. It is clear that using such type of methodologies is not the good choice.

The opposite decision may go to “Do not use a formal methodology anymore”. Wait, be careful, you are opening the door to the “Caw Boy Coders” age to return, in which each programmer create his own software in the way that he can do. If you did and did not notice the chaos that surrounds you, just wait till the big disasters wake you up to know that you had taken the wrong decision.

Now, it is evident that you need a simple methodology that you can live with to reach your goals in an efficient and economic manner.

“eXtreme Programming, XP” can be the genius solution in your case because of:

- Simplicity and easy communication between team members.
- Enable the extreme utilization of individual programmer creativity.
- Suitability for the small teams, typically between 2 to 12 members.
- Suitability for the highly changeable requirement environments or when the customer does not have a firm idea on his requirements (i.e. cases in which, dynamically

1 RUP has been enhanced so that it meets the real-world needs of typical organizations. The first enhance was the expansion of the scope of RUP to include the entire software process, not just the development process. The second one was adding the support for the management of project life cycle. The enhanced version is called the "Enterprise Unified Process (EUP)".[1]

changing requirements is the only constant).

- Ability to manage projects with high risk involved.
- The evidence impact on teams and individuals productivity

eXtreme Programming is classified as one of the most agile, elegant and light weight methodologies that have only a few rules, practices and documents that are easy and simple to follow correctly.
It is based on a contradiction with the old believe, which was saying: ("Software which is engineered to be simple and elegant is less valuable than software that is complex and hard to maintain"). The consideration of this idea to be a big myth originated eXtreme Programming.

The reason for XP's success is its focus on customer satisfaction. The methodology is designed to deliver the software that your customer needs when it is needed. XP had been built on four basic concepts: (1) Simplicity, (2) Decomposition, (3) feedback and (4) Iterations. Simplicity is achieved by containing only a little number of rules which is essential for the production process. So, it is suitable for SMEs.

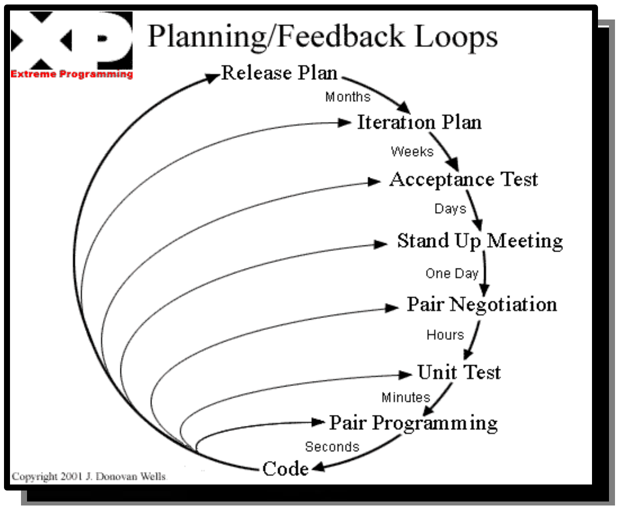
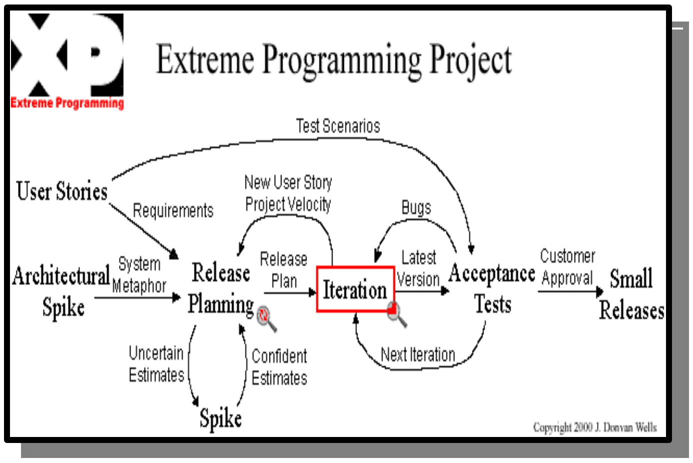


Fig (1): Planning Loops

The Decomposition of the software production process into smaller and smaller number of activities is the main issue in XP, figure (1) shows how the project release is divided into smaller nested loops.

Fast feedback from each iteration output - in each loop - provides the "Quality Assurance Group[2]" with the required information to take the needed corrective action, because it is simpler to calculate the error between the desired and actual performance in closed loop systems than in open ones. Finally, in SMEs – with highly dynamic environment -, it is essential to go throw an iterative production life cycle (composite of nested iterations); XP provides this iteration with an adjusting factor called the "Project velocity[3]".



2 Quality assurance (QA) is an essential part of the XP process. On some projects QA is done by a separate group, while on others QA will be an integrated into the development team itself. In either case XP requires development to have much closer relationship with QA.

3 The project velocity (or just velocity) is a measure of how much work is getting done on your project. To measure the project velocity you simply add up the estimates of the user stories that were finished during the iteration. It's just that simple. You also total up the estimates for the tasks finished during the iteration. Both of these measurements are used for iteration planning.

Fig (2): XP Programming Project

Now, let us walkthrough the process to see why it is different from the traditional heavy weight methodologies. XP journey starts with the collection of what we call "User Stories", which is considered to be similar to the "Use Cases" in "Unified Modeling Language UML", but it is different. It is typically written by the customer (i.e. as usage scenarios). Completely simple and techno-syntax free, and also uses as a basis for time estimation and acceptance test. Another input to the release planning is the "spikes", which is a very simple program to explore potential solutions and reducing risks. Now, you are ready to conduct the "Release Planning". Release planning includes: (1) How many releases? (2) Who do what, when and how long questions' answers.
Releases (usually have monthly time) are divided into reasonable number of iterations (usually in weeks) that takes the allocated user stories, velocity correction and bugs from the previous one as inputs. The key features in XP appear inside iterations, such as: (1) Doing acceptance test each few days. (2) stand up meeting every day (even in coffee breaks). (3) Peer reviews (after few hours of work). (4) Finally, the peer programming, this is the bright start concept in XP. Just wait….!, two programmers on one PC are not a waste of human resources, but it is a time and effort saving. It saves time by finding the solution faster (two brains are always better than one). It also saves efforts of errors correction by detecting and solving errors early, just as soon as it emerges.

Pair programming also provides what we can call "Collective Code Ownership", which protects you from "coding bottle necks" (i.e. single owner), especially in an environment with a highly turnover rates. Before we finish, it is clear that iteration need a carefully planning to conduct all these activities effectively and efficiently.

"Project Planning" process area in the XP project depends on dividing the project into releases, iterations and then user stories; finally each story is divided into a reasonable number of tasks. Usually, plans fail due to three common factors: (1) the false assumption that tasks are independent. (2) Lateness is passed down the schedule; earliness is not. (3) The Student Syndrome, which refer to the tendency not to start tasks earlier than necessary. XP successfully give solutions to these three factors. For the first, it assumes the dependency between tasks and performs the estimation on a story instead of a task. For the second, it hides the complete plan from the developers so, they usually act as fast as possible. For the third, XP provides "stand up meeting" and "peer programming". XP needs a computerized tool to help in planning and tracking. Traditional tools such as "MS Project" can help, but it is more helpful to use a specially designed tool for XP. The "XPlanner" is a good typical example.

It must be clear that XP methodology does not address the topic of project or process documentation. But it is simply integrated with what we call an "Agile Modeling, AM".

Finally, XP is not the magic solution. It is important to notice that it needs a good training and experience to use. If you try to use the XP to the first time, it is highly recommended to first conduct a pilot test on a moderate risky project. If you become familiar with it, you can widely use and enjoy results.

**References**
[1] http://www.extremeprogramming.org/
[2] http://www.userstories.com/
[3] http://www.xprogramming.com/
[3] “User Stories Applied for Agile Software Development”, a book by Mike Cohn.

All web sites are taken as a reference and checked for last update at (16 Feb, 2004).

**Biography**
Eng/ Ahmed S. El-Shikh is the Quality Assurance Manager in the “Systems Advisory & Solutions, (SAS)” company. His interests include software engineering aspects, software quality management, statistical quality control and process Improvement approaches specially the “Six Sigma, DMAIC”. Feedback, comments and questions are appreciated by the author; author contact: (el_shikh@sas-sys.com) Feedback, comments and questions are appreciated; please contact me on: (el_shikh@sas-sys.com) xcv specially the “Six Sigma, DMAIC”.

# Online Training and Knowledge Repositories for today's Software Engineers

***By Dr. Hoda M. Hosny***

The continued advancement in software development processes and methods is fundamental to Egypt's ongoing efforts towards leadership in the software industry within the region. These efforts have so far been very well initiated by government, industry and university working together. It is now time for them to focus on the significant advancement in the use of the best software processes, through electronic training and online repositories, to produce the highest quality software. The result will not only strengthen our industrial and research capabilities, but will also lead to innovations that provide broader access to education and training for students and practitioners alike, at least on the local level.

With the speed of change in technologies, lifelong learning on the part of today's software developers is inevitable. Moreover, developers from different backgrounds should be able to work and collaborate together (even remotely) and take advantage of the benefits offered by the available technology when moving from the general to the integrated software development techniques.

On-the-job training of practicing software engineers, through electronic means, and the availability of sources of knowledge when they are most needed should not be far fetched.

The suggestion here is for founding a continued training framework in which coordinated efforts between industrial, academic and government participants to foster the training of best Software Engineering (SE) practices, accreditation standards and professional upgrade on a national level, is made possible. Such a framework would assume electronically-based training programs and content material drawn from international curricula as well as successful practices and experiences from industry partners.

There is undoubtedly a joint responsibility between academia and industry which lies in shaping these professionals who can deliver dependable solutions to societal problems. All relevant stakeholders (from industry, academia and society) along with their unique needs and constraints, can and should participate.

On the international level, initiatives were taken in recent years to increase collaborative partnerships between industry and academia to foster the professional development of university students and practicing software engineers such as in [2] and [5]. Others, such as the ISEUC (International Software Engineering University Consortium), a joint effort between 35 or more institutes of higher learning from some of the world leading countries in the industry [3], provide courses (non-degree granting) to students globally via distance learning for the same purpose.

Repositories of shared practices and knowledge represented by case studies and other means were also made available in collaboration with industry.

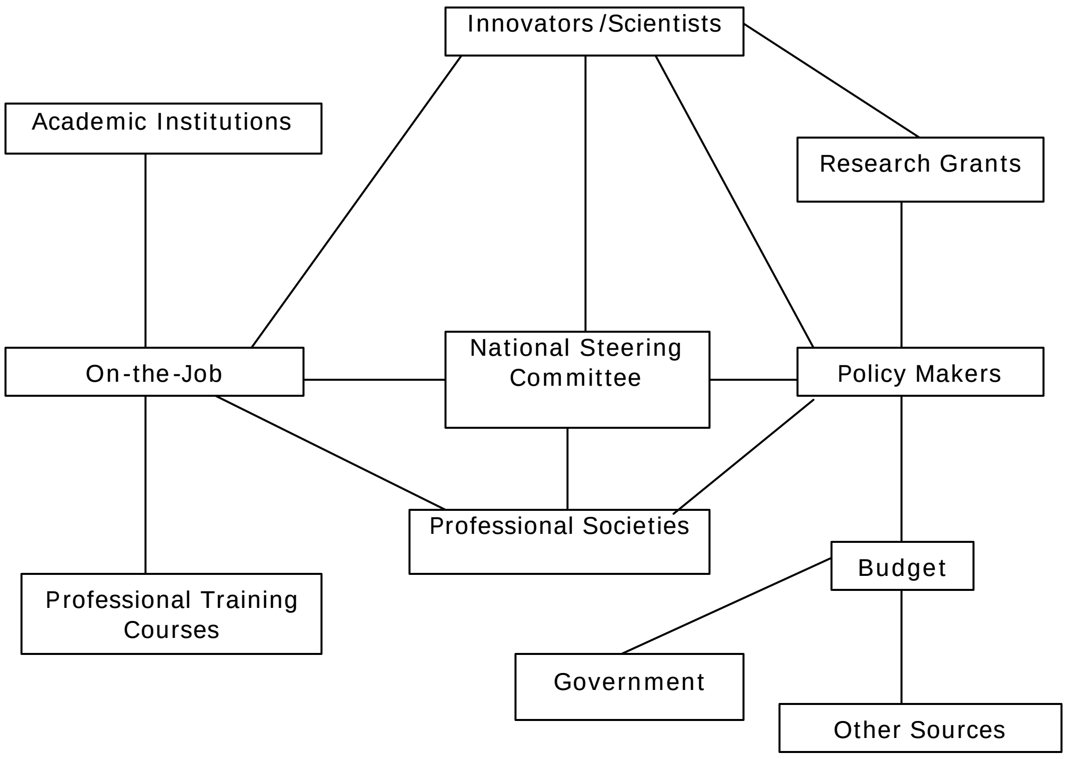


***Fig. 1 Partner Elements in the Continued Training of Software Engineers***

Fig. 1 highlights the collaborated partner elements that must be involved in the continued training of software engineers [1]. The double-edged arrows between the major parties are there to suggest the constant flow of cooperation between these partner elements and the changes in their existing roles as they interact with one another. The changing industry needs, curricular responses as well as funding and support and the role of government must all be taken into consideration while planning, developing and implementing training programs at such a professional level. It is inevitable to formulate national goals and objectives and to interpret them in terms of curricular content and approaches. In formulating such goals there are many clear trends in national development which have a strong scientific or technological content and which are an inherent part of the structure of a modern nation state. This is true whether the economy of such a state is highly developed or developing. The objective is to bring together members of the academic and industry communities to discuss the nature of their interactions and what roles government, society, academic and research institutions should play in fostering university-industry ties and in advancing the profession. Among the issues that need to be addressed are how do we best train software engineers for the future ? How must the university/industry organize itself to facilitate this training ? what is the appropriate infrastructure for modern training ? How can all this fit within the international standards ? The globalization of the economy and of the work force as well as student movement across countries emphasize the need for a common basis for accreditation criteria, policies and guidelines [4].

In conclusion, a framework for administering online training programs and repositories of knowledge for software engineering practices may be handled through a joint collaboration between industry, government and academic institutions. The recipients will undoubtedly appreciate the opportunity to receive just-in-time training and/or knowledge that is individualized for each user, is highly interactive, and is available any time any place when it is most needed.

## References

[1] Hosny, Hoda M., “Towards a Collaborated E-Training Program for Software Engineers”, Proceedings of the 2nd E-learning Conference held at AUC, January 10-11, 2004.

[2] Kornecki, A.J., Khajenoori, S. Gluch, D. and Kameli, N. “On a Partnership between software Industry and Academia”, proceedings of the 16th conference on Software Engineering Education and Training (CSEET03), IEEE press 2003.

[3] Modesitt, K.L. “International Software Engineering University Consortium (ISEUC): A Glimpse into the Future of University and Industry Collaboration”, in the proceedings of the 15th conference on Software Engineering Education and Training (CSEET02), IEEE press 2002.

[4] Sobel, A. (Moderator), Panel Discussion entitled “ SWEBOK as an International Foundation for Software Engineering Program Accreditation” in the proceedings of the 14th conference on Software Engineering Education and Training (CSEET’01), IEEE press 2001

[5] Strooper, P., Carrington, D. Newby, S. and Stevenson, T. “Teaching Software Engineering Fundamentals to Practicing Engineers”, proceedings of the 16th conference on Software Engineering Education and Training (CSEET03), IEEE press 2003.

# Software Testing Activities Should Start Early in the Development Lifecycle

**By: Madiha A. Hassan**

There are many perceptions and misconceptions of what software testing is. Some people see software testing as something developers do as they write code or once code is complete. Others believe that testing should be performed along-side development activities.

It is a well-established fact that the sooner a defect is found the less expensive it is to fix. Consider the definition: "Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results".

Testing the application throughout its development lifecycle, emphasizing the early lifecycle activities helps increase the overall effectiveness and efficiency of this effort. Further, early involvement allows evaluation of important planning, design and development decisions with respect to how these decisions aid or impair the testability of the application.

**Software Development Life Cycle & Testing Activities.**

Test resources have at least one associated activity within any phase of a project lifecycle regardless of the development methodology followed:

***Project Planning*** - The project plan should be associated with the test plan which schedule testing activities. As well as, assigning resources to these tasks. This will allow any potential resource allocation problems to be raised at the beginning of the project while there is still time to do something about any shortfall.

***Requirements and Specifications*** - While the requirements are being gathered and written, Testing Team begins writing a Test Strategy to outline the overall approach for testing and related assumptions and constraints. Testing Team should also review the requirements for clarity, completeness, ambiguities, and testability. This means preventing defects from occurring in the code and having to be found during a future testing phase, when it is more expensive to fix.

***Design*** - While Development team is documenting the architecture and detailed design for the application, Testing team continues Requirements Analysis and begins creating Test Cases (for inclusion in the Test Plan) and any other Test Assets (test coverage matrices, test data, test automation frameworks, etc.). Testing team also reviews the design documents for accuracy and testability which will improve quality of the product while saving time and money.

***Coding (unit testing )*** - Once coding begins, Testing team must work towards completing Test Cases, Test Plans and Test Assets. Testing team begins performing unit testing until error free status.

***Integration & System testing***: Testing team will then move on to high-priority Test Cases in areas where sufficient development has occurred. As integration progresses, Testing team should be able to test more and more of the application's functionality, reporting all defects. When the application nears functional completeness, Testing team completes one or more test passes (execution of all Test Cases). Finally, focus shifts to system, scenario, stress, and performance testing . which must be more effective as the process started early.

***Release -*** Testing team can sometimes be responsible for the final builds that lead up to the release. When the release decision is made, Testing team should collect and archive all project data, documentation, source and tools. Testing team also participates in the project Post Mortem, where problems and solutions encountered in a project are reviewed for learning purposes.

***Support and Maintenance*** - As externally-found defects are reported to the support personnel, defect reports are analyzed by Testing team to determine if the defect is reproducible and whether it was known prior to release or not.
Starting test activities early means you can catch small quality problems before they become big quality problems later on. Review plans and designs for weaknesses before the software is ready to be tested. Test the software as soon as it is available and log those bugs all the way through the project.

# What testing techniques software engineers should know?

***By Omar Kamal***

Software projects depend heavily on human-beings to develop and engineer software products. Through-out the software lifecycle engineers introduce enormous amount of errors. Errors may be an outcome of incomplete requirements analysis process, poor design process, or inefficient communication process. That is why testing the software product at different phases is necessary to both verify and validate the software product against its specifications

A lot of efforts are being spent to demonstrate how to manage and improve the software testing process. Those improvement strategies often bring up a number of discussions at the engineering level. What are the essential testing techniques needed for realizing any improvement project? Which techniques are the best fit for the kind of software projects the organization are engaged in? What is black-box testing and what is white-box testing strategy? This article is explaining abstractly the meaning of both the black-box and white-box testing strategies

Beginning by asking ourselves what is the exact purpose for software testing and what the detailed objectives are? Is it to demonstrate that the software works, or is it to demonstrate that the software doesn't work? Or is to reduce the software product failure risk perception to acceptable values"
Boris Beizer in his great book "Software Engineering Techniques" discusses the purpose of testing in a very interesting manner and captures the change in the view for the purpose of software testing. He concludes by stating that "the above goals for testing are cumulative goals and even the most testable software must be debugged, must work and must be hard to break

By knowing the test objectives, the task turns to be selecting the strategy that offers the means to meets those objectives

**Black-BoxTesting**
Also known as "data-driven" or "input/output-driven" testing, this strategy is based on the concept of viewing the software product as a black box. In other words, the way the the software transforms the inputs into outputs are not examined. The code should behave according to the specifications. That is why the test data are derived from the specifications. In this type of testing the software tester is taking the user's point of view

Test cases are developed to answer one or all of the following questions

- Does the program meet its specifications"
- How to test system behavior and performance"
- What are input test vectors that excite the system or transfer it to different observable states"
- How are the boundaries of a data class isolated"
- How to measure input data sensitivity"
- What is the system behavior under different capacity, volume, size, or scale"

**Black Box Exhaustive input testing**
It is impractical if not impossible to test the software product exhaustively. A huge number of test cases should be developed and executed to exercise different combinations of valid inputs.

This may take months or even years to generate and execute which makes it unrealistic except for exceptional trivial cases. Not only this, you may need to develop another set of in-valid inputs to make sure the program won't fail if it receives such un-expected inputs

***For example***

Let us assume that we are planning to test a program that computes z where ***z = ax + by +cz***. such that a, b, c are defined constants and x,y,z are input variables. Now suppose for simplicity that x,y,z are integers only valid from 1 to 40. The number of all possible combinations is 40*40*40 = 64000. If it takes around 2 minutes to develop, document, and execute a test case, we need more than 2 months to finish testing. This is for a simple equation with 3 bounded finite input variables and linearly associated output. What would be the case if there are much more variables with complex non linear relations? The answer is, the exhaustive input testing is impossible to realize even for fast computers working for years

**White Box Testing**

Also known as 'Structural testing", "Logic-driven testing" or "Glass-box testing", this strategy dose looks at the implementation details. The internal structure of the software, the procedural design, the programming style, the database design and other coding details are at the center of focus for such testing strategy. The way the overall software is decomposed can influence the testing process. What can be functional tested at a certain layer can be structurally tested at a different layer. Test cases are developed with one or all of the following motives

- To execute every independent path from entry to exit
- To execute all statements in all independent paths at least once
- To execute every branch with all its sub-branches combinations at least once
- To execute loops at boundary conditions and carefully selected typical values

**White Box Exhaustive path testing**

Executing all paths may seem to be the solution at the first glance but thinking deeply we can easily figure out that this is not true. Executing every path in the program doesn't indicate that the program is behaving according to its specification. A program that executes each and every path successfully and provides the user with tallest student instead of the shortest one as written in its specifications is simply a wrong program. In addition, missing paths are not discovered by exhaustive path testing. To cover every path in the program that includes very small loops may not be a difficult task, but what about normal size loops, big loops, or even nested loops. Running an exhaustive path testing for a program that has any of those loops may be unrealistic if not impossible

So, both the black box exhaustive input testing and the white box exhaustive path testing are not realistic if not impossible. Hence, the question becomes "Is there any sensible, wise, and economic testing strategy that both demonstrates program workability and reduces our perception of program failure objectively[3]

**Combine both techniques**

Combining black-box testing techniques with white-box testing techniques results in a hybrid testing strategy that hopefully may answer the previous question. This is not a silver bullet, software designers should be aware when developing their design that there are no reasons for punishing software testers intentionally or un-intentionally by significantly reducing the software testability to achieve little design gain . If testability is not clearly realized in the design and the code, implementing any testing technique may realy be a hard task

The following list includes some but not all of the black-box and white-box techniques. A detailed explanation of those techniques is out of scope of this article. More details can be found on references mentioned at the end of the article

**Control flow graphs**

*Flow graphs can be used to represent control flow in a program. Using these graphs, different paths can be visualized and test case development becomes easier. Control flow is different than flow charts: it is more abstract and aggregates non-branching statements into single nodes*

**Random testing**

*Attempt testing in a haphazard way, entering data randomly until the system fails. It is likely to uncover some errors in the system, but it is very unlikely to find them all.*

**Field testing**

*It is an actual target environment testing at customer-like environment, where collecting real data is the purpose. It is necessary to demonstrate system capabilities for acceptance.*

**Logic Coverage Testing**

- *Statement: each statement executed at least once*
- *Branch: each branch traversed (and every entry point taken) at least once*
- *Condition: each condition True at least once and False at least once*
- *Branch/Condition: both Branch and Condition coverage achieved*
- *Compound Condition: all feasible combinations of condition values at every branch statement covered (and every entry point taken*
- *Path: all feasible program paths traversed at least once.*

*Various strategies have been developed for identifying useful subsets of paths for testing when Path Coverage is impractical: Loop coverage, Basis paths coverage, Transaction flow coverage and Dataflow coverage.*

**Equivalence Partitioning**

- *This technique is also known as input space partitioning.*
- *Its idea is to partition the input space into a number of equivalent classes. Each class repesents a category of inputs such that one could expect, based on the specification, that every element of a certain class would be transformed (i.e., mapped to an output) in the same manner (either correctly or incorrectly.)*
- *An example of simple categorization is valid and in-vaild classes*

**Boundary Value Analysis**

*A technique based on identifying, and generating test cases to explore boundary conditions. The probability of find errors at extreme values or at the boundaries is higher than typical values. The technique may be applied to both input and output conditions.*

**Error Guessing**

- *Also known as Ad Hoc Testing, Artistic Testing, etc...*
- *Testers utilize intuition and experience to Identify potential errors and design test cases to reveal them.*

**Cause-effect analysis**

- *Cause-Effect Analysis is an approach for developing test cases to cover different combinations of input ''Causes'' that produce output ''Effects.'' A cause may be thought of as a specific input condition, or an ''equivalence class'' of input conditions*

- *An effect may be thought of as a specific output condition, or an observed change in a program state.*

**Mutation Analysis**

*The purpose of this technique is to measure test cases set sufficiency.*

1. *A program that successfully passed certain set of test cases, is intentionally doped with minor defects.*
2. *The modified program is tested using the same set of test cases.*
3. *The more the test case catches those inserted defects the more the test case is adequate.*
4. *The process also provides guidance in the creation of new test cases to provide more adequate testing.*

**Invest in education**

Educating software test engineers in techniques listed above is essential and very important factor in improving any software test process. At the end of the day, testing should reduce the perceived software failure risk to acceptable values. Team leaders and senior test engineers should select objectively testing techniques that best-fit their software product according to the project context and technology. Software coders and designers should at least go through orientation sessions that cover those techniques in-order to (realize) testability. Having such knowledge is useful even if it is not directly used by software testers. Knowing that "X" testing tool generates control flow diagrams, and support domain testing versus "Y" testing tool that don't support any of them, but support syntax testing is beneficial in selecting the appropriate testing tool.

**References**

1. Roger Pressman, Software Engineering, A Practitioner's Approach 5th Edition, McGraw-Hill, 2001.
2. Beizer, B., Software Testing Techniques 2nd Edition, Van Nostrand-Reinhold, 1990.
3. Myers, G., the Art of Software Testing. Wiley, 1979.
4. www.cis.ufl.edu/class/cen5035/notes/test.pdf

**Biography**

Omar Kamal, 7 years of experience in wireless telecommunications software development, training, and software quality management. Working as a senior software engineer in QuickTel Research and Development Used to work with Lucent Technologies, Hewlett Packard, and Eitisalat. Holds a bachelor's degree in telecommunications engineering from Cairo University, and master's degree in business administration from City University. Also a Certified Quality Manager by the American Society for Quality

# Software Process Improvements (SPI) at ITSoft, Cairo

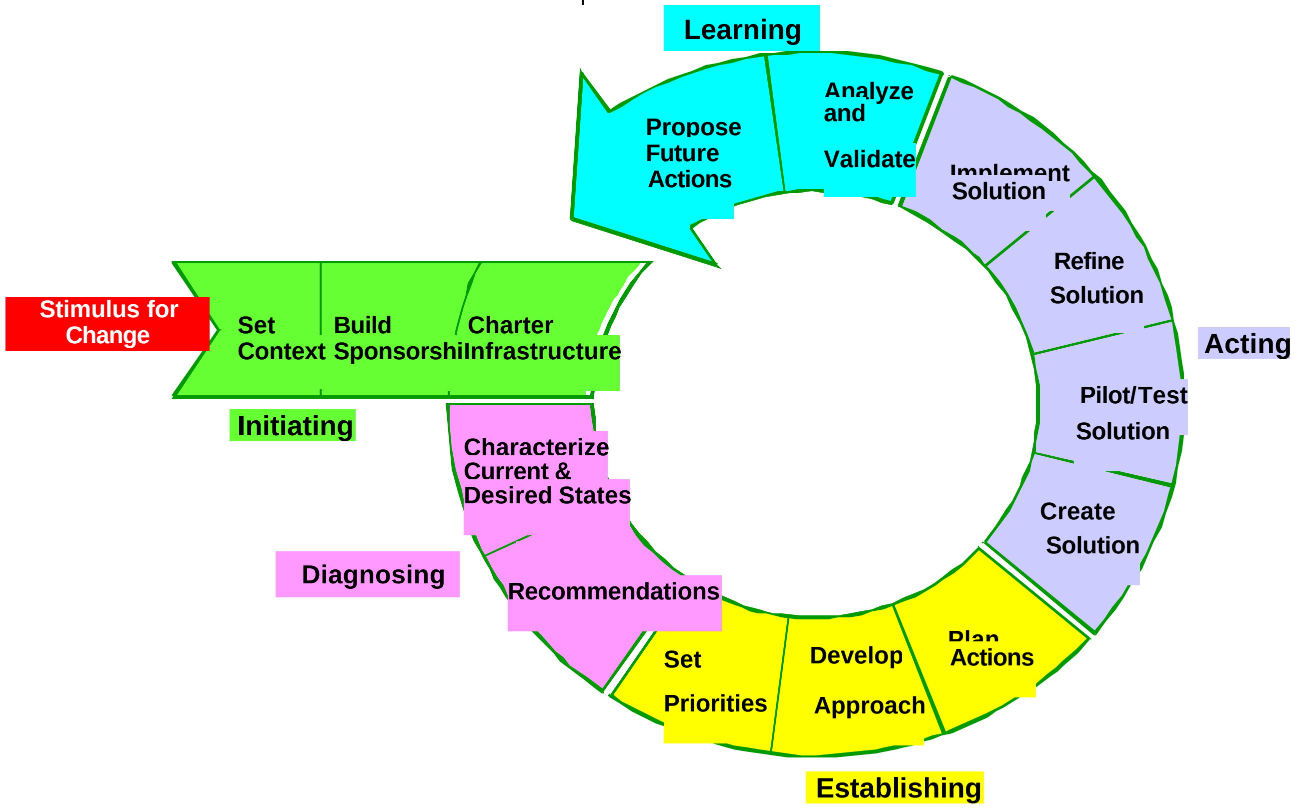
***By: Sameh S. Zeid***

**Background on ITSoft**
ITSoft is an Egyptian organization located in Nasr City Free Zone, specialized in Off-shore software services. ITSoft is structured around its product lines and have a development center for each. All ITSoft services are directed towards non-local clients making ITSoft facing direct competition with other International Software organizations, especially from India. In year 2000, ITSoft has obtained its ISO 9001:2000/ Tick IT certification

**Introduction**
The Capability Maturity Model (CMM®) from the Software Engineering Institute (SEI) is a world recognized accreditation. The CMM® provides organizations with a reference model for leveraging their software practices while being recognized with International clients

Being in International competition, quality of services and products has become a prerequisite for the existence of ITSoft. Driving from the tenet that product quality is bound by effectiveness of supporting processes, SPI is not an option

**ITSoft Approach for SPI**
ITSoft has adopted the IDEAL® approach as its methodology for SPI in a customized form. The same phases of IDEAL were applied but rather in a modified flavor that is tailored to ITSoft



**The phases of the SPI are explained in the following sections.**

**Phase 1: Initiation**

During the Initiation Phase, Process Improvement was chartered as a project, while formally designating the Project Manager. The charter authorized formation of SEPG, with pre-defined responsibilities
Sponsoring of the SPI project is practiced by ITSoft top management.
The Context set for the project was as follows

- Core team was designated for daily project work and follow-up
- Identified and acknowledged the various training requirements
- Retain the help of an external consultant
- Organized main processes to be used for managing the SPI project, this included SEPG Guidelines, Release Process and Communication

**Phase 2: Diagnosing**

This phase is where more information is collected for detailed project planning. During this phase, the following two major milestones were addressed

Gap Analysis (mini assessment)

- Select sample projects from every product line to be assessed against CMM® KPAs. Knowing our current maturity level by that time was a key for deciding our action plans and areas of focus
- Assessed existing QMS: Being an ISO company, ITSoft had its own QMS. This QMS has been assessed with respect to the requirements of CMM® and for its general suitability to existing business practices

Decide on next actions
Based on the outcome from Gap Analysis milestone, the SPI core team has recommended the following

- Software Processes that need to be developed.
- Select projects that will pilot the implementation of these processes.
- Prioritize the implementation of various processes.

**Phase 3: Establishing**

This was the major phase of the SPI project where main bulk of resources was used. The milestones of this phase are described in the following paragraphs.

1. Process Development

A process is unique to the organization. Though considering what others have done is useful, an organization should develop its own process representing its special characteristics which is normally unique. Process Development is iterative process that required several reviews and involved training to process owners

2. Agree on key concepts

- Certain key concepts that were not addressed adequately have been emphasized and taught to various process owners
- Defining the prime contractor vs. end client
- Separate QA from QC. QC has been moved to be part of the product line.
- Enforce that quality of a work product is the responsibility of the producer.
- Establish a layer of Project Software Managers (PSM) solely accountable on project deliverables.
- Form pool of auditors from practitioners after providing them formal training.

3. Construct Software Process Database (SPDB)

The SPDB holds organization process assets and can be accessed by all individuals at ITSoft. The SPDB is owned by SEPG and guidelines have been released for its maintenance

4. Develop Organization-wide Software Life Cycle (SLC)

ITSoft is comprised of various development centers serving different partners with each having different standards and SLC. Having organization-wide SLC has served ITSoft unifying the development standards across various groups and to have ground for sharing information across various groups. It served to ensure that best practices are manifested in the organization-wide SLC. However, the implementation of SLC was different from one product line to another

5. Software Size Estimation

ITSoft has adopted Functional Point as its main standard for estimating Software Size. With respect to Size Estimation, ITSoft has categorized its projects to: Development, Product Customization, Maintenance and Localization. Specific guidelines were developed for every project category

6. Measurements & Metrics

ITSoft has developed a process for collecting measurements, calculate and analyze metrics. Metrics were developed covering the areas of Defects, Project Planning & Tracking, Requirements Management, SQA and Process Efforts. Time sheets activities have been updated to reflect the new process introduced. To reduce efforts involved in collecting Measurements, the existing forms used by projects were utilized after doing minor modifications

7. Establish Quality Organization

ITSoft has established a Quality Organization that supported its SPI. The new organization was formed of:

- SEPG: Maintaining QMS and facilitate SPI initiatives;
- QAG: Perform various types of Auditing;
- Requirements Control Board: for approving any external commitment; and
- Management Review committee: Monthly review of Quality activities, major non-conformities, and opportunity for improvement.

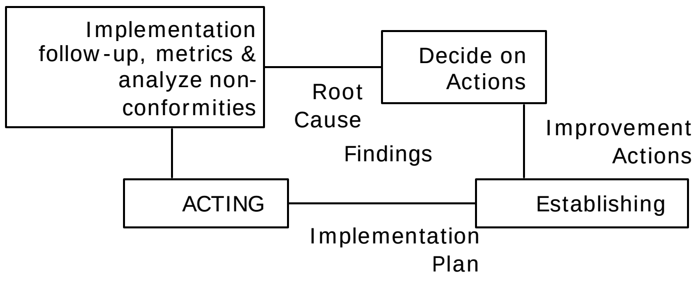
**Phase 4: Acting**

This was the phase were actual implementation of various processes have taken place. Some of key activities may be summarized as the following:

- Start by implementing Client Engagement & Project Planning in various projects
  - o External commitment management
  - o Project Plan and Status Report can make quick advantage
  - o Project Plan and subsidiary plans are PSM ownership
- Have Process Facilitation through Project
- Implement Tracking
  - o Team Activity reports
  - o Project Management Review
  - o Tracking Schedule, Effort, Defects and Risks
- SQA is frequent activity and a key for getting implementation feedback.

**Phase 5: Learning**

Please refer to the following diagram showing how SPI cycle was done at ACTING phase.



Implementation follow-up, metrics and analysis of non-conformities were the basis to locate Root Causes. Actions were planned to remedy the Root Causes. Findings derived from ACTING were the main input for analysis and to feedback into the SPI cycle.

**Summary of Lessons Learned**

Though ITSoft has not yet achieved the formal CMM® accreditation, the lessons learned so far from this project may be summarized as:

- Make sure that people understand the difference between documentation and "process".
- Is what you really want a certification or tangible improvement?
- There are hundreds of CMM® complaint processes, but yet every organization has its own.
- CMM® model is NOT meant to teach you back-ground.
- Test and SQA should be separated.
- To manage between commercial and SPI efforts we need to have core team for SPI.
- Though process compliance is the target, Templates can really help.
- Maintenance is a "Project".
- Previous experience in SPI Pays-Off

**Terms**

| | |
|---|---|
| QMS | Quality Management System |
| KPA | Key Process Area |
| CMM | Capability Maturity Model for Software version 1.1 |
| SEI | Software Engineering Institute |
| SPI | Software Process Improvement |
| DOD | Department of Defense of the US |
| IDEAL | Standard methodology from SEI for Software Process Improvement |
| Process Owner | Practitioners who are involved in the implementation of a process |
| RCB | Requirements Control Board |

**References**

- Capability Maturity Model for Software version 1.1 (CMU/SEI-93—TR-024).
- IDEAL: A User's Guide for Software Process Improvement (CMU/SEI-96-HB-001).

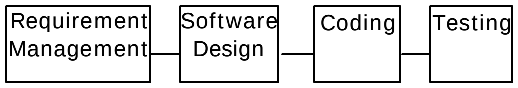
**Biography**

Sameh S. Zeid, PMP has been involved with software development and management since 1985. Sameh played various roles on the technical and managerial levels for establishing core business solutions. Sameh has a B.Sc. degree in Computer Science from Faculty of Engineering and Post-Graduate degree in Operations Research.

# Towards an Effective Testing Process

**By Nermine Faltas**

In a typical environment and with all the challenges faced during software development, testing comes in a later stage in the software lifecycle. As time to market is one of the main factors for success, testing is pushed to a later stage as the development cycle goes fast and phases overlaps to meet the market expectations. Furthermore, as the development or implementation phase gets late or a bit longer, it squeezes the testing time. So, the testing team faces a tough situation where they have to first understand the application they are testing, plan for the test execution, develop test cases, execute them and at the end perform regression testing to make sure that all issues are resolved. To illustrate the typical lifecycle, the following is a waterfall lifecycle model:



For a company that is fast growing, Time to market is not the main success factor anymore. Product quality appears to be a great challenge as well. So, any software development firm begins to think of what are the considerations that it should take to have effective framework in place for planning, executing, and managing the testing process. This article explains the main considerations to be thought of in order to achieve a proper and effective framework.

## Dedicated Test Engineer

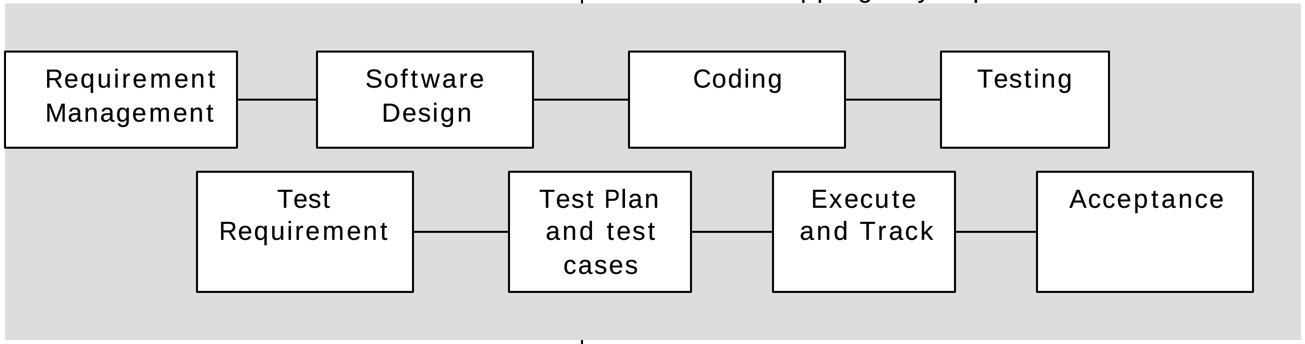
When someone begins to mention dedicated test engineer, management starts to think about budget or cost of extra resources. However, dedicated test engineer means a person who looks to the product with an eye different from the development team. This does not require extra resources, many alternatives can happen. Scenario one is to train some developers on the testing process and techniques and they can act as a test engineer in projects he is not involved in as a part time. The important issue in this scenario is to have a clear process, clear activities and some checklists to support developers. Another scenario is to have one resource taken out of the development team and act as a full time test engineer. This scenario is very useful too. This person can be senior enough to contribute to define the testing process and testing approach. The best scenario in a fast growing company would be to have a dedicated team with a manager driving the process and tracking test engineer performance and improving their skills. Experience shows that a test engineer who is looking to a product from its quality point of view and not aware of all product design can detect defects faster and will be able to drive the quality of the product to meet the customer and the user experience.

Test Engineers can detect and correct defects early enough and with minimum cost. So, the return on investment will be as high as it minimizes cost of rework over time. For the dedicated test engineer to have an effect on the product quality, His role and activities all through the lifecycle have to be very clear.

## Testing engaged early in the lifecycle

As every software project starts with a requirement phase, test engineers should be engaged directly after the requirement phase or in a late stage in the requirement where there is a solid documentation, he can study. They should be provided with the application requirements up front, so they can develop system test cases that cover both stated functional and non-functional requirements. Doing so, they can build a test plan with those test cases in order to validate the product later in test execution. And, as they execute those test cases, they could track any Issues they find in the application and send those back to development for resolution.

The test team should be able to clearly associate test cases to requirements, so that if a requirement changes, it is clear that associated test cases should also change.



The advantages of such a system are clear

- Testing team will have a better understanding of application requirements and can create more effective test plans
- Problems can be identified earlier in the process
- Test cases can be based on actual application requirements.

## 3. Checklists & Support Tools

### *Checklists*

To efficiently carry out the test process, there should be checklist identifying the aspects to be tested in every type of test conducted for the product to guide test engineers on all part they should take care of.
A simple example for testing a form will illustrate this point. In the user interface testing one of the test types to be conducted which is testing a simple form requires checking the form from different aspects. One aspect is the form display like field's alignment, fields spacing, etc…. Another aspect is the form functionality which consists of testing mandatory fields and the system reaction if they are empty, invalid data in fields… Another aspect is error messages.
So, every time the interface of form is tested, same aspects should be tested over and over. So, it is better to have sample checklists with possible test consideration to be done on a certain test to optimize testing cycle and eliminate skipping any aspect

### *Support Tools*

Another aspect is to have supporting tools to make the testing process more efficient. Tools will have proper insight on reporting the testing status. Also, they will help in having statistical data from which you can evaluate the product state and detect decisions about releasing the product or not. The main tools that should be there are:

- Test case management
- Defect Tracking tools

*Test Case management*
Test cases should be documented in an organized manner for every test type. Recommended fields for a test case would be:

- The test type
- The test case number
- Conditions

- o Flow
- o Expected results
- o Actual Result
- o Status

Test cases should be logged in a tool or an excel sheet with some calculations for multiple reasons. First one is to have proper reporting on the number of test cases conducted, number of test cases passed and number of test cases failed. Second, is to promote reusability of test cases. In the Internet environment for example a registration component, where a user can register to a site, change your profile information…, exists in pretty all of the applications. So, re-writing test cases for every function or looking in documents about the test cases would be a headache. However, test cases can be taken as is from a sheet or the tool and executed again to optimize testing time and minimize skipping any test case. There are many free test case management tools to begin with and start practicing logging and getting reports fro m the tool

*Defect tracking tool*
Defects should be logged into a tool as this will be shared between the test engineer and the developers working on the project. A test engineer who found a defect, will access the defect management tool to add this defect. The suggested fields for a defect would be:

- o Project Name
- o Defect type
- o Description
- o Flow done
- o Associated test case
- o Developer responsible
- o Area where the defect was found
- o Status of the defect
- o Found in which build
- o Solved in which build

From those data the process flow for a defect would be easy in the communication and knowledge to solve this defect in a minimum time.

It would also make sure that no defect would slip without being resolved and re-checked by the test engineer. Also, performing some trends analysis on the defect sources would be useful in determining where the common areas of defects are and the corresponding defect density. Furthermore, it would be an add-on for analysis and determining improvement opportunities to minimize the defect rate

## 4. Conclusion
This article is meant to identify how a company can start to initiate some activities to change its testing process to reach an effective process for a better quality for the product. This article does not mention any testing techniques or testing approaches. It did not mention how we can conduct test planning or how to design test cases. All of the above are details that can be obtained from various sources like stickyminds.com which is one of the best resources in testing. A test process should be reached and get repeatable in multiple projects to be generalized across the organization.

## References

- o www.empirix.com, Implementation of web application testing process white paper
- o IMPLEMENTING AN EFFECTIVE TEST-MANAGEMENT PROCESS, A MERCURY INTERACTIVE WHITE PAPER
- o QA Concepts and Implementation Guidelines, August 1, 2000 Nadeem Kayani
- o http://www.stickyminds.com

**Bibliography**
Nermine Faltas holds a Bachelor of Science in Computer Science from the American University in Cairo in 1995. She has been working for LINKdotNET for the past 6 years starting her carrier as internet application developer, then as a senior internet developer. She received training on project management and she worked as a project manager for two years. She is currently working as a quality manager in LINKdotNET. She is responsible for product testing and process quality assurance. Nermine's current interest is software quality management. She has been working on all process improvement activities and product quality since