Nonlinear and Quantitative Software Engineering Method  
Based on Complexity Science  

Jay Xiong   Lin Li  
NSEsoftware, LLC., USA  
www.nsesoftware.com  
jay@nSESsoftware.com  lilin@nSESsoftware.com  

Abstract—This paper introduces a nonlinear and quantitative software engineering method (NSE-Method), which is the core part of the Nonlinear Software Engineering paradigm (NSE) based on complexity science. NSE-Method offers a nonlinear, holistic, global, and quantitative approach for software development. The objective of NSE-Method is to make it possible to help software development organizations double their productivity, halve their cost, and improve the quality of their software products by several orders of magnitude simultaneously. NSE-Method has been successfully implemented and supported by the NSE- Method platform, Panorama++, integrated from many automated software engineering tools.

Keywords: Software engineering method; complexity science; nonlinear software engineering; software engineering paradigm  

I. INTRODUCTION  

Today software has become the driving force for the development of all kinds of sciences, engineering, and businesses. As pointed out by David Rice, “Like cement, software is everywhere in modern civilization. Software is in your mobile phone, in your home computer, in cars, airplanes, hospitals, businesses, public utilities, financial systems, and national defense systems.” [1]  

But unfortunately, software itself is not well engineered, many critical issues (low quality and productivity, and high cost and risk, etc.) have existed with software development for more than 50 years.

The root cause is that software is a nonlinear system where a small change may bring big impacts to the entire system — the “Butterfly-Effect” — but the old-established software engineering paradigm with all existing software development methods is an outcome of linear thinking, reductionism, and the superposition principle that the whole of a nonlinear system is the sum of its parts, so that with it almost all software engineering activities are performed linearly, partially, locally, and qualitatively.

For efficiently resolving those critical issues, a new software engineering method called NSE-Method has been innovated which complies with the essential principles of complexity science. Theoretical comparison with initial applications shows that compared with the old software engineering methods it is possible for NSE-Method and the corresponding tools to help software development organizations double their productivity, halve their cost, and improve the quality of their software products by several orders of magnitude simultaneously.

II. CRITICAL ISSUES ADDRESSED  
The critical issues addressed by NSE-Method have existed with software development for more than 50 years, including (a) Incomplete – the process is incomplete: “The Unified Process suffers from several weaknesses. First, it is only a development process… it misses the concept of maintenance and support…. The relative software investment that most organizations make is allocating roughly 20% of the software budget for new development, and 80% to maintenance and support efforts.” (Scott Ambler, [2]). In fact not only RUP, but almost all existing software development methods do not really support software maintenance without various bidirectional traceabilities.

(b) Inconsistent - the design documents and the source code of a software product developed using existing methods are inconsistent with each other after code modification again and again.

(c) Unreliable - “Major software projects have been troubling business activities for more than 50 years. Of any known business activity, software projects have the highest probability of being cancelled or delayed. Once delivered, these projects display excessive error quantities and low levels of reliability.” (Capers Jones, [3]). This is why software disasters happen often.

(d) Un-maintainable - “Over three decades ago, software maintenance was characterized as an ‘iceberg’. We hope that what is immediately visible is all there is to it, but we know that an enormous mass of potential problems and cost lies under the surface. In the early 1970s, the maintenance iceberg was big enough to sink an aircraft carrier. Today, it could easily sink the entire navy!” (Roger S. Pressman, [4]). “The fundamental problem with program maintenance is that fixing a defect has a substantial (20–50%) chance of introducing another”(Brooks, [5]). This is why so many cloud computing system failures were reported in 2011, including

* Sony’s Playstation Network (4/21/2011)  
* Amazon Web Services (4/21/2011)  
* Twitter Service (2/25/2011)  
* Netflix Streaming Service (3/22/2011)
* Intuit Service and Quickbooks (3/28/2011), and more [6].

(e) **Invisible** – the existing visualization methods, techniques, and tools do not offer the capability to make the entire software development process and the work products visible; in most cases, the generated charts and diagrams are used for modeling only, and are not holistic and not traceable.

(f) **More** – there are more critical issues existing with software development, including low quality and productivity, high cost and risk, etc.

III. **Why have those critical issues existed for more than 50 years without being resolved?**

As we know well, many software engineering methods and technologies have been innovated and applied by software scientists, experts, and engineers such as the Object-Oriented software development approach, the platform-independent Java programming language, the model-driven software development methods, and the Agile software development approaches, but why those critical issues have existed for more than 50 years without being resolved?

Before answering this question, let us consider what makes those critical issues exist:

(a) **The process models** – they are linear ones, no matter if it is a waterfall-like model, an incremental development model which is “a series of Waterfalls”, or an iterative development model in which each time of the iteration is a waterfall, or a new process model recommended by Alistair Cockburn to combine both Incremental and Iterative development together [7] shown in Fig. 1, with which there is no upstream movement at all, the work flow is always going forward from the upper phases to the lower phases. The result is that defects introduced in the upper phases easily propagate to the lower phases to make the defect removal cost increase as much as tenfold.

(b) **The software development methods** – as shown in Fig. 2, they are one-way software development methods (Top-Down or Bottom-Up) based on linear thinking, reductionism, and Constructive Holism principle that the software components are developed first, then the system of a software product is built through the integration of the components developed (see Fig. 3) - “Assemble the product from the product components, ensure the product, as integrated, functions properly and deliver the product.” [8], so that system testing is performed after all components have been developed. With these methods, the quality of a software product is carried out through inspection and software testing after production – it is too late.

(c) **The software testing paradigm** – most software defects are introduced into a software product in the requirement development phase and the product design phase as shown in Fig. 5, but the existing software testing paradigm can only be dynamically used after production as shown in Fig. 6 and Fig. 7, so that NIST (National Institute of Standards and Technology) concluded that “Briefly, experience in testing software and systems has shown that testing to high degrees of security and reliability is from a practical perspective not possible. Thus, one needs to build security, reliability, and other aspects into the system design itself and perform a security fault analysis on the...

(j) The entire software engineering paradigm - it is an outcome of linear thinking, reductionism, and the superposition principle that the whole of a software system is the sum of its parts, so that with it almost all software development activities are performed linearly, partially, locally, and qualitatively. It offers linear approaches to handle the development of a software product, a nonlinear system. One of the primary reasons that many businesses fail is an attempt to resolve the issues of a non-linear system with a linear process.

IV. HARDEST CHALLENGES

As described above, almost all components of the old-established software engineering paradigm makes bad "contributions" to make those critical issues (low quality and productivity, and high cost and risk, etc.) exist!

According to complexity science, the behaviors and characteristics of the whole of a complex system emerge from the interaction of its all components, and cannot be inferred simply from the behavior of its any individual part, so that only improving one or a few components such as focusing on the improvement of the software engineering process and the software maintenance process only will not be able to make significant improvement to the whole of the software engineering paradigm – it could be the main reason why the failure rate of the implementation of CMM/CMMI is about 70% [9]. “There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.”[5], so that for efficiently resolving those critical issues and slaying software “werewolves” (a monster of missed schedules, blown budgets, and flawed products) [5], we need a qualified "Silver Bullet" - a complete revolution in software engineering.

What does a revolution mean? According to Thomas Kuhn’s book (1962) “The Structure of Scientific Revolutions”, a revolution means a drastic, complete, and fundamental change of paradigm to resolve some outstanding and generally recognized problem that can be met in no other way.

V. THE SOLUTION: NSE-METHOD

For efficiently resolving those critical issues with software development, a new software engineering method, NSE-Method, is innovated and successfully implemented, which complies with the essential principles of complexity science, including the Nonlinearity principle, the Holism principle, the Complexity Arises From Simple Rules principle, the Initial Condition Sensitivity principle, the Sensitivity to Change principle, the Dynamics principle, the Openness principle, the Self-organization principle, and the Self-adaptation principle.

(a) Facts and some experts’ conclusion on software engineering

• Mc Cracken/Jackson: Life-Cycle Concept Considered Harmful. “System requirement can never be stated
fully in advance, not even in principle, because the user doesn’t know them in advance - not even in principle”.

- Balzer: Specification and Implementation are Intertwined. “In actual practice developments steps are not refinements of the original specification, but instead redefine the specification itself ... there is a much more intertwined relationship between specification and implementation than the standard rhetoric would have us believe.”

- Rich/Waters: “Writing a complete specification in a general-purpose specification language is seldom easier and often incredibly harder than writing a program. Furthermore, there has been little success in developing automatic systems that compile efficient programs form specifications”.

- Demillo, Perles, Lipton: “If a formal program is transformed from an informal specification then the transformation process itself must necessarily be informal ... in the end, the program itself is the only complete and accurate description of what the program will do”.

- Joseph Barjis: “Despite diligent efforts made by the software engineering community, the failure of software projects keeps increasing at an alarming rate. After two decades of this problem reoccurring, one of the leading causes for the high failure rate is still poor process modeling (requirements specification). Therefore both researchers and practitioners recognize the importance of business process modeling in understanding and designing accurate software systems. However, lack of direct model checking (verification) feature is one of the main shortcomings in conventional process modeling methods.”

- Frederick P. Brooks Jr.: “There has to be upstream movement... experience and ideas from each downstream part of the construction process must leap upstream, sometimes more than one stage, and affect the upstream activity.”

- Frederick P. Brooks Jr.: “The fundamental problem with program maintenance is that fixing a defect has a substantial (20-50 percent) chance of introducing another. … Clearly, methods of designing programs so as to eliminate or at least illuminate side effects can have an immense payoff in maintenance costs:” , “Plan the System for Change”, “Testing the Specification”, “Add one component at a time” “Incremental development – grow, not build, software”, “To keep documentation maintained, it is crucial that it be incorporated in the source program, rather than kept as a separate document.” , “People Are Everything (Well, Almost Everything).”, “A program should be shipped with a few test cases, some for valid input data, some for borderline data, and some for invalid data”. 

- Cockburn: “The fundamental characteristics of ‘people’ have a first-order effect on software development”.

- Sneed H.: “Diagrams are not always the best means of modeling a solution. A solution can also be described in words. The important thing is that one model is enough – either the code or the diagrams. They should be reproducible from one another”

- Adam Rin, Father of CA-IDEAL 4GL: ”Production and testing is a multi step process with CASE, first you do the design, then the program generation, then the compilation, then a link edit, and then you test. If an error occurs, it occurs in the program and not in the design. To correct it, you have to start over again from the top.”.

- Capers Jones: “Defect prevention plus inspections and tests give highest cumulative efficiency and best economics.”

Those experts’ conclusions have been reflected in NSE-Method and its innovation process.

(b) The objective of NSE-Method

As shown in Fig. 8, with the existing software development methods, software quality, productivity, and cost strongly affect each other. For instance, improving the quality will reduce the productivity or increase the cost, so that it may not bring extra benefit to software development organizations.

![Fig. 8 The relationship among quality, productivity, cost, and risk with today’s software development methods](image)

<table>
<thead>
<tr>
<th>Quality</th>
<th>Productivity</th>
<th>Cost</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↓</td>
</tr>
</tbody>
</table>

Fig. 8 The relationship among quality, productivity, cost, and risk with today’s software development methods

Differently, the objectives of NSE-Method is to make it possible to help software development organizations double their productivity, halve their cost, and improve the quality of their software products by several orders of magnitude simultaneously as shown in Fig. 9.

![Fig. 9 The objectives of NSE-Method](image)

NSE’s Objectives

Quality → Risk (removing 99.99% or more of defects through defect prevention and defect propagation prevention)

Productivity → Cost (Double (reducing about 2/3 of the effort and cost spent in maintenance and defect prevention, etc.)

(c) The foundation of NSE-Method

The foundation of NSE-Method is complexity science which is considered as the greatest science research
achievement after Relativity and Quantum Mechanics. Complexity science explains how holism emerges in a nonlinear system, and more. Definitions of complexity are often related to a complex system with many parts that interact to produce results that cannot be explained by simply specifying the role of each part. This concept contrasts with traditional Newtonian constructs, which assume that all parts of a system can be known through analysis, detailed planning can produce predictable results, and that information flows along a predetermined path.

(d) The Framework for the Innovation of NSE-Method

It seems that no successful “killer application” of complexity science has been reported yet – Why? In our opinion, the major reason is that the “Sunlight” of complexity science cannot directly “Reach” the target without removing the big “Umbrella” in the middle – the old-established paradigm such as the software engineering paradigm consisting of many parts including the process models, the development methods, the modeling approaches, the test paradigm, the SQA paradigm, the documentation paradigm, the maintenance paradigm, and the project management paradigm – as mentioned above, all of them are outcomes of linear thinking, reductionism, and the superposition principle, each of them offers bad “Contributions” to make the critical issues (low quality and productivity, and high cost and risk) exist for more than 50 years, so that for efficiently solving the problems in a nonlinear system, it is recommended that the problem is handled in two major steps: the first one is to complete the paradigm shift by the organization performing the tasks (or a tool vendor), then the second one is to handle the detailed tasks of a nonlinear system by applying the corresponding new paradigm established in the first step. Thus, a general paradigm-shift framework called FDS (Five-Dimensional Structure Synthesis method) was innovated. There are five axes representing the five elements in FDS: People/Logic, Environment, Process Phases, New Method/Paradigm, and Principles of Complexity Science. FDS not only can be used for the software industry, but also can be used in other industries. Fig. 10 shows that FDS is used for the innovation of NSE-Method. When FDS is used for paradigm shift or new method innovation, it is required to comply with the essential principles of complexity science, so that a waterfall-like method or process model will not be established because it does not comply with the Nonlinearity principle and the Holism principle of complexity science.

(e) The Family Members of NSE-Method

As shown in Fig. 11, NSE-Method mainly consists of ten members:

(1) **NSE Visualization Method** - it shifts the foundation of visualization from reductionism to complexity science to make the entire software development processes and the work products visible by automatically generating holistic, graphical, interactive, dynamic, virtual, and traceable diagrams directly from the source code of stub programs or regular programs, making a software product much easy to understand, test, and maintain. An application example is shown in Fig. 12.

(2) **NSE Testing Method** - it shifts the foundation of software testing from reductionism to complexity science through the innovative Transparent-Box testing method (Fig. 13) which seamlessly combines functional testing and structural testing together: to each set of inputs used to test an entire software product being developed, Transparent-Box testing method and the corresponding tool not only checks whether the output (if any, can be none when this method is dynamically applied in software modeling to test requirement specifications) is the same as what is expected, but also helps users to check whether the real program execution path covers the expected path specified in control flow, and then automatically establishes the bi-directional traceability among the related models, documents, test cases, and the entire source code according to the description part of the test cases, so that this method can be dynamically used in the entire software development process to find functional defects, structural defects (with capability to highlight untested branches and conditions graphically), and inconsistency defects. Fig. 14 shows an application example.
Fig. 13 Transparent-Box testing method

Fig. 14 An application example of transparent-box testing: a bug found even if the output is the same as what is expected (this defect comes from that a "break" statement is missing, so that the result "4" is produced through 2 times 2 rather than 2 plus 2)

Fig. 15 shows that the Transparent-box testing method and the corresponding tools support MC/DC (Modified Condition/Decision Coverage) test coverage measurement quantitatively with the diagramming capability to highlight un-tested conditions graphically.

Fig. 15 Quantitative MC/DC test coverage measurement

(3) NSE Traceability Establishment Method - it shifts the foundation of software traceability from reductionism to complexity science to establish accurate and precise traceability among the requirements specifications, documents, test cases, and the source code of an entire software product holistically and dynamically through (a) Time Tags automatically inserted into the description part of test cases and the code test coverage database to accurately and precisely map the test cases and the tested source code together, and (2) some keywords (such as @WORLD@, @PDF@, @HTML@, @EXCEL@, and @BAT@) to indicate the format of a related document, followed by the file path, and a bookmark for opening the traced document from the specified location as shown in Fig. 16. Some bi-directional traceability application examples are shown in Fig. 17. More application examples are provided in the sections of (4) NSE Modeling Method and (10) NSE Maintenance Method. This kind of bi-directional traceability is not only automated, but also self-maintainable: no matter if the test cases or the source code is changed, after re-running the test cases, new Time Tags will be automatically generated to map them together again - this kind of traceability is the base for software quality assurance driven by defect prevention, and software maintenance performed holistically, globally, and quantitatively with side-effect prevention in the implementation of requirement changes or code modifications.

Fig. 16 NSE Traceability establishment method

Fig. 17 Application examples of the bi-directional traceability

(4) NSE Modeling Method - it shifts the foundation of software modeling from reductionism to complexity science to make the obtained holistic and dynamic models/diagrams traceable for static defect removal, executable indirectly for debugging, and dynamically testable for dynamic defect removal. As shown in Fig. 18 NSE Modeling Method uses source code of stub programs (in forward engineering) or regular programs (in reverse engineering) written in a platform-independent programming language such as Java or a target platform-dependent language to automatically enable it as both a design specification language (without changing the language itself) to automatically generate NSE diagrams which are holistic, interactive, virtual, dynamic, traceable, and indirectly (through the source code) or directly (by playing the captured GUI operations back) executable for high-level system abstraction used for human understanding of a software product, and a programming language using the source code.
for computer understanding of the software product as shown in Fig. 19. Fig. 20 shows the notations for Actor and Action and the stub source code. Fig. 21 shows the mapping between Use Cases and NSE J-Chart notations and why Use Case is not automatable but the corresponding J-Chart notations are automatable. Because source code is the only source for NSE modeling, it enables modeling to become pre-implementation, and implementation to become further modeling non-linearly - Top-down and Bottom-up.

With NSE Modeling Method, the resulting models (diagrams) are dynamically testable for dynamic defect removal using the Transparent-box testing method, even if there is no real output. For instance, from the beginning there is only a stub program written in C++ - the main() function listed as follows:

```c++
void main(int argc, char** argv) {
    int key;
    if(argc==1 /* Missing a parameter */ || argc > 2 /* Having an extra parameter */) {
        cout << "Invalid Commands: \n" << argv;
    } else {
        if(strcmp(argv[1],"New_Order")==0 )
            New_Order();
        cout << "*** A_New_Order () called. ***\n";
    }
}
```

One test case for Transparent-box testing is written as follows:

```
# test case 1 for New Order
# @HTML@ C:\Billing_and_Payment10\Requirement_specification.htm#New_Order
# @WORD@ C:\Billing_and_Payment10\Prototype_design.doc bmname New_Order
# @WORD@ C:\Billing_and_Payment10\TestRequirements.doc bmname New_Order
# [/path] main(int, char**) {s0, s1, s9}  [/path]
# Expected output : none
```

After the execution, two defects are found as shown in Fig. 22.

Fig. 22 Two defects found: the execution path didn’t cover the expected one; one bookmark used to open a document is wrong.

One of the defect comes from a extra space:

```c++
if(argc==1 /* Missing a parameter */ || argc > 2 /* Having an extra parameter */) {
    cout << "Invalid Commands: \n" << argv;
} else {
    if(strcmp(argv[1],"New_Order")==0 )
        New_Order();
    cout << "*** A_New_Order () called. ***\n";
}
```

Another defect comes from incorrect location of the bookmark "New_Order". Fig. 23 shows the two defects have been removed.

Fig. 23 New result after the two defects have been removed

(5) **NSE Design Method** - it shifts the foundation of software
design from reductionism to complexity science and the Generative Holism Principle that the whole of a complex system will exist (as an embryo) earlier than its components, then grows up with its components incrementally to make all versions (including the stub system) executable for customer testing to get feedback to improve the design. In this way design becomes pre-coding - coding can be done by directly editing the stub modules from the designed call graph, and coding becomes further design by rebuilding the database to automatically update the design (Fig. 24).

Fig. 24 Design and coding with NSE-Method

(6) **NSE Coding Method** - with NSE-Method coding is performed incrementally through bottom-up coding ordering on the designed system call graph. In this way unit testing and integration testing and system-testing can be combined together without using stub modules to replace the modules called by a module being coded and tested. If the module relationship is changed after incremental coding, the design will be automatically updated by rebuilding the database (see Fig. 25 and Fig. 26).

Fig. 25 Design becomes pre-coding

Fig. 26 Coding becomes further design

(7) **NSE Quality Assurance Method** - it brings revolutionary changes to software quality assurance through defect prevention, testing using the Transparent-box method, and inspection using traceable documents and source code from the first place down to the retirement of a software product to meet Edwards Deming’s product quality principle.

(8) **NSE Documentation Method** - it brings revolutionary changes to software documentation to automatically generate holistic, dynamic, virtual documents traceable to the source code to keep them always consistent with each other.

(9) **NSE Project Management Method** - it combines the software project management process and product development process together to make the project management documents such as the project development plan and schedule charts traceable with the requirements and the source code for finding problems early and fixing the problems in time. Fig. 27 shows a project schedule chart is traced from one of the related requirement. Fig. 28 shows how to balance the development of two related projects.

Fig. 27 Making the project management documents traceable with the requirements

Fig. 28 Multiple Project Management

(10) **NSE Maintenance Method** - it brings revolutionary changes to software maintenance to make software maintenance be performed nonlinearly, holistically, globally, and quantitatively with side-effect prevention in the implementation of requirement changes or code modifications supported by various bi-directional traceabilities as shown in Fig. 29 for the implementation of a requirement (Addition) change.

Fig. 29 (a) Performing forward tracing through the main test cases related
With nonlinear and quantitative NSE-Method, the quality of a software product is ensured from the first place through defect prevention, dynamic testing using Transparent-box testing method, and inspection using traceable documents and source code, so that defects propagated to maintenance will be greatly reduced, plus that software maintenance is performed nonlinearly, holistically, globally, and quantitatively to prevent the side-effects in the implementation of requirement changes and code modifications, so that the effort and cost spent in software maintenance will be almost the same as the effort and cost spent for a (new) software development (about 25% of the total effort and cost) - it means that about half of the total effort and cost can be saved - equal to double the productivity and halve the total cost as shown in Fig. 30.

With nonlinear and quantitative NSE-Method, the quality of a software product is ensured from the first place through defect prevention, dynamic testing using Transparent-box testing method, and inspection using traceable documents and source code, so that defects propagated to maintenance will be greatly reduced, plus that software maintenance is performed nonlinearly, holistically, globally, and quantitatively to prevent the side-effects in the implementation of requirement changes and code modifications, so that the effort and cost spent in software maintenance will be almost the same as the effort and cost spent for a (new) software development (about 25% of the total effort and cost) - it means that about half of the total effort and cost can be saved - equal to double the productivity and halve the total cost as shown in Fig. 30.

With nonlinear and quantitative NSE-Method, the quality of a software product is ensured from the first place through defect prevention, dynamic testing using Transparent-box testing method, and inspection using traceable documents and source code, so that defects propagated to maintenance will be greatly reduced, plus that software maintenance is performed nonlinearly, holistically, globally, and quantitatively to prevent the side-effects in the implementation of requirement changes and code modifications, so that the effort and cost spent in software maintenance will be almost the same as the effort and cost spent for a (new) software development (about 25% of the total effort and cost) - it means that about half of the total effort and cost can be saved - equal to double the productivity and halve the total cost as shown in Fig. 30.
edits the stubs to make design becoming pre-coding using **NSE Coding Method**

(d) Incremental unit testing combined with integration testing and system testing using **NSE Testing Method**

(e) Quality assurance using **NSE Quality Assurance Method** through defect prevention, inspection using traceable documents and the source code, and dynamic testing

(d) When the module relationship is changed after incremental coding, update the design results by re-building the database to make coding becoming further design.

(e) Making the project development process and the work products visible using **NSE Visualization Method**

(f) Automatically generating various documents using **NSE Documentation Method**, and make the documents traceable to the source code

(g) Automatically establishing the traceability among models/documents (including project management documents and reports) and test cases and source code through test case execution using **NSE Traceability Establishment Method**

(h) Using the bi-directional traceability established to validate the requirements often to find possible defects (including project management issues) and resole them in time

(j) Responding to requirement changes or code modification in real time with side-effect prevention in the implementation of the changes using **NSE Maintenance Method**

(k) If a solution method for implementing a requirement is not satisfied, go back to step (1) to perform prototyping and review again through backward traceability - implementation becomes further modeling.

(l) Combining project management process and project development process together using **NSE Project Management Method**

(l) Frequently delivering working product versions to customers for feedback to improve the product.

(7) With NSE-Method before the product retirement, all software engineering tasks should be performed continuously.

VI. MAJOR FEATURES OF NSE-METHOD

The major features of NSE-METHOD include:

(a) Based on complexity science; (b) Nonlinear - Top-down plus Bottom-up; (c) Holistic; (d) Dynamic; (e) Quantitative.

VII APPLICATIONS OF NSE-METHOD

NSE-Method can be used for

(1) A new software development

(2) A software product being developed using other methods (in this case users only need to re-write the test cases and set bookmarks for the related documents, other work can be done automatically by NSE support platform Panorama++)

(3) For a legacy software product, because the source code is the source for automatic software modeling and documents generation, source code itself is not the best documents, but is the best source for automatically generating various documents.

All screenshots provided in this paper come from real applications.

VIII WHAT IS SOFTWARE?

Software: “instructions (computer programs) that when executed provide desired features, function, and performance; data structures that enable the programs to adequately manipulate information; and documents that describe the operation and use of the programs”[4]. But, with NSE, software is re-defined as: a computer program; the data used; all of the related documents (including the test case scripts) traceable to and from the source code: the database built though static and dynamic measurement of the program; plus a set of Assisted Online Agents for handling the complexity and supporting the testability, visibility, changeability, conformity, reliability, traceability, and maintainability - making the software products adaptive and truly maintainable in the new working environment at the customer site, and that the acceptance testing can be done dynamically in an automated way with mouse clicks only (for more detailed information, see the NSE-Book[10]).

VIII CONCLUSION

Software is a nonlinear system where a small change may bring big impacts to the entire software product.

"The next century will be the century of complexity" (Stephen Hawking, January 2000) - Complexity science is the study of nonlinear, dynamic systems and the process of self-organization. It is a field derived from multiple disciplines—physics, chemistry, biology, and mathematics - we propose that the general theory for software engineering should be complexity science! The disciplines should be complied with by software engineering are the essential principles of complexity science, including the Nonlinearity principle, the Holism principle, the Initial Condition Sensitivity principle, the Sensitivity to Change principle, the Dynamics principle, the Openness principle, the Self-organization principle, and the Self-adaptation principle, which have been successfully complied with by NSE-Method.

Linear, partial, local, and qualitative software engineering era has ended; nonlinear, holistic, global, and quantitative software engineering has come of age!

REFERENCES


