General Comparison between the Old-Established Software Engineering Paradigm and NSE (Nonlinear Software Engineering Paradigm)

Jay Xiong
NSEsolution, LLC., USA
jayxiong@yeah.net, jay@nsesolution.com

Abstract
This paper describes the major differences between the old-established software engineering paradigm and a new software engineering paradigm called NSE (Nonlinear software engineering paradigm based on complexity science) in detail. The essential difference between them is how to handle the relationship between the whole and its parts of a software system. The former adheres to the reductionism principle and superposition principle that the whole is the sum of its parts, so that nearly all software development tasks/activities are performed partially and locally, such as the implementation of requirement changes. The latter complies with the Holism Principle of complexity science, that a software product is a Complex Adaptive System having multiple interacting agents (components), of which the overall behavior and characteristics cannot be inferred simply from the behavior of its individual agents but emerge from the interaction of its parts, so that with NSE nearly all software development tasks/activities are performed globally and holistically to prevent defects in the entire software lifecycle.

Keywords: Silver Bullet, software engineering paradigm, modeling testing, quality assurance, maintenance

1. Introduction
Low quality, productivity, and project success rate, and high cost and risk are the critical issues which have existed with the old-established software engineering paradigm for more than 40 years. The root cause is that software is a nonlinear system where a small change may bring big impact to the entire system – the “Butterfly-Effect”, but the old-established software engineering paradigm is an outcome of 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, and locally.


This paper will further compare the differences between the old-established software engineering paradigm and NSE in almost all parts, including the modeling approaches, the software development methods, the software development processes, the testing paradigms, the quality assurance paradigms, the documentation paradigms, the visualization paradigms, the maintenance paradigms, and the project management paradigms.

2. General Comparison between the Old-Established Software Engineering Paradigm and NSE

2.1 Software Definition

A. The software definition of the old-established software engineering paradigm

Software is defined as
* 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 [3].

B. The software definition of NSE

Software is defined as
* 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 (including the test case script files too); plus
* the database built through static and dynamic measurement of the programs; and
* a set of Associated Online Agents (AOA, automated and intelligence tools working with the database) for supporting testability, reliability, visibility,
changeability, conformity, and traceability to make the software program maintainable, adaptive, and that the static and dynamic measurement results can be viewed easily, and the acceptance testing can be dynamically done in a fully automated way through mouse clicks only.

2.2 Software Engineering foundation

A. The old-established software engineering paradigm

The old-established software engineering foundation is based on linear thinking, reductionism, and the superposition principle that the whole of a system is the sum of its parts, so that with it almost all software development tasks/activities are performed linearly, partially, and locally.

B. NSE

The NSE foundation is based on complexity science with a set of essential principles including the Nonlinearity principle, the Holism principle that a whole is greater than the sum of its parts - the characters and the behavior of a complex system is an emergent property of the interactions of its components (agents), the Dynamics principle, the Self-organization principle, the Self-adaptation principle, the Openness principle, the Initial Condition Sensitivity principle, the Sensitivity to Change principle, the Complexity Arises From Simple Rules principle, etc., so that with NSE, almost all tasks/activities are performed globally and holistically through a nonlinear process.

2.3 Software Development Methods

A. The old-established software engineering paradigm

A Top-Down or Bottom-Up method is used linearly.

B. NSE

A nonlinear Top-Down plus Bottom-up method is used, driven by defect prevention supported by various traceability (see Fig. 1).

2.4 Software Engineering Process Models

A. The old-established software engineering paradigm

With the old-established software engineering paradigm all software engineering process models are linear with no upstream movement at all as shown in Fig 2.

B. NSE

The NSE foundation is based on complexity science with a set of essential principles including the Nonlinearity principle, the Holism principle that a whole is greater than the sum of its parts - the characters and the behavior of a complex system is an emergent property of the interactions of its components (agents), the Dynamics principle, the Self-organization principle, the Self-adaptation principle, the Openness principle, the Initial Condition Sensitivity principle, the Sensitivity to Change principle, the Complexity Arises From Simple Rules principle, etc., so that with NSE, almost all tasks/activities are performed globally and holistically through a nonlinear process.

2.3 Software Development Methods

A. The old-established software engineering paradigm

A Top-Down or Bottom-Up method is used linearly.

B. NSE

A nonlinear Top-Down plus Bottom-up method is used, driven by defect prevention supported by various traceability (see Fig. 1).

2.4 Software Engineering Process Models

A. The old-established software engineering paradigm

With the old-established software engineering paradigm all software engineering process models are linear with no upstream movement at all as shown in Fig 2.

B. NSE

The NSE foundation is based on complexity science with a set of essential principles including the Nonlinearity principle, the Holism principle that a whole is greater than the sum of its parts - the characters and the behavior of a complex system is an emergent property of the interactions of its components (agents), the Dynamics principle, the Self-organization principle, the Self-adaptation principle, the Openness principle, the Initial Condition Sensitivity principle, the Sensitivity to Change principle, the Complexity Arises From Simple Rules principle, etc., so that with NSE, almost all tasks/activities are performed globally and holistically through a nonlinear process.

Fig. 1 NSE Software development method

Fig. 2 Various Linear Software Engineering Process Models with No Upstream Movement at All
B. NSE

With NSE a nonlinear software engineering process model is offered as shown in Fig. 3.

Fig. 3 NSE software engineering process model

The major features of NSE process model include:
* **Dual-process**: NSE model consists of the pre-process and the main process. They are different but also closely linked together.
* **Nonlinear**: The NSE model is nonlinear, complying with the Nonlinearity principle and the Holism principle.
* **Parallel with Multiple tracks**: “Much of software architecture, implementation, and realization can proceed in parallel.” [4]. For reducing waiting time and speeding up software development processes, the NSE process model supports tasks being performed in parallel with multiple tracks through bidirectional traceability.
* **Real time**: “Timely updating is of critical importance.” [4]
* **Incremental development with two-way iteration**: The NSE process model supports incremental development with two-way iteration, including refactoring to handle highly complex modules and performance bottlenecks with side-effect prevention. When a critical issue is found in the main process, the work flow may go back to the preprocess for selecting a better solution method, and so on.
* **The software development process and software maintenance process are combined together seamlessly**: With the NSE process model, there is no big difference between the software development process and the software maintenance process – both support requirement changes through side-effect prevention.
* **The software development process and the project management process are combined together closely**: all documents including the project management documents such as the project development plan, the schedule chart, and the cost estimation report are traceable with the requirement implementation and the source code for better control of the product development. NSE process model also supports the critical requirements and most important requirements being implemented early with the assigned priority to avoid budget overuse – if necessary, some optional requirements and not so important requirements can be ignored temporarily.
* **Adaptation focused rather than predictability focused**: the entire world is always changing, so the NSE process model is adaptation focused rather than predictability focused – it supports requirement changes, code modifications, data modifications, and document modifications to make them consistent and updated with side-effect prevention.
* **Defect prevention driven**: **People are considered as the first order driver for software development** - When people consider “people as the first-order” to software development, they focus on how to trust and support people better for their jobs, but ignore the other side of people’s effect on software development – almost all defects introduced into software products are made by people, the developers and the customers. So NSE supports people in two ways: one is to support them with better methodology, technology, and tools; another one is to prevent the possible defects to be introduced into the software products by people - it is done mainly through various automated and bidirectional traceabilities.

2.5 Software Modeling Approaches
A. The old-established software engineering paradigm

Offering linear, partial, and local software modeling approaches through two kinds of sources with one in graphics drawn by hands or using a graphic editor for human understanding of a software product, and another one in test format for computer understanding of the software product – there is a big gap between the two kinds of sources as shown in Fig. 4.

![Fig. 4 Existing software modeling approaches](image)

B. NSE

Offering nonlinear, holistic, and global software modeling approach (called NSM – Nonlinear Software Modeling approach) using one kind of source (the source code of a stub program using dummy modules having an empty body or only some function call statements without detailed program logic, or the source code of a regular in forward engineering or reverse engineering) for both human understanding of a software system using better graphics automatically generated from the source code for high-level system abstraction, and computer understanding of the software system using the source code directly as shown in Fig. 5.

![Fig. 5 NSE software modeling approach](image)

The major features of NSE software modeling approach include:

1. NSM is based on complexity science, complying with the essential principles of complexity science, particularly the Nonlinearity Principle, the Dynamic Principle, and the Holism Principle, so that with NSM almost all software modeling and engineering activities are performed nonlinearly, holistically, and globally, rather than linearly, partially, and locally.

2. NSM uses one kind source (the source code of a platform-independent programming language (such as Java/Java-DLI) or even a platform-dependent programming language) for human understanding of a complex software product through the colorful and meaningful Models/Diagrams automatically generated from the source code for high-level abstraction, and computer understanding of the complex software product using the source code or the transformed source code, so that with NSM the models/diagrams are always consistent with the source code.

3. NSM makes design become pre-coding, and coding become further design – offering Top-Down plus Bottom-Up software development approach.

4. NSM offers dynamic software modeling approach rather than static one: (a) with NSM, the generated models/diagrams are existing dynamically - when a chart or a diagram is shown, the corresponding generator is always working for users’ commands to operate to meet users’ needs through the interface – using the chart/diagram itself; (b) with NSM, the generated models/diagrams are dynamically executable through the corresponding source code; (c) with NSM, the generated models/diagrams are dynamically traceable to the requirements and source code.

5. NSM completely solves the inconsistency issues between the generated models/diagrams and the source code.

6. NSM brings revolutionary changes to software modeling quality by making the generated models/diagrams traceable for static defect removal, and executable through the corresponding source code for dynamic defect removal.

7. The models/diagrams generated with NSM are accurate and precise to the source code.

8. With NSM a software product developed through nonlinear software modeling and engineering is much easier to understand, review, change, test, and maintain.

2.6 Software Testing Paradigm

A. The old-established software engineering paradigm

Functional testing and structural testing are separated, performed after coding, can not be used to find defects in
B. NSE

Functional testing and structural testing are combined together seamlessly using the innovated Transparent-box testing method which not only checks whether the output (if any, can be none) from a software product being developed is the same as what is expected, but also helps users to check whether the real program execution path covers the expected program execution path, then automatically establishes bi-directional traceability among related documents and test cases and the source code according to the description of the test cases through Time Tags (when a test case is executed) and some special keywords (see Fig. 6 and Fig. 7).

![Transparent-box for finding defects in five ways](image)

Fig. 6 The innovated Transparent-box testing method

Software quality assurance is mainly based on inspection and software testing after coding, violates W. Edwards Deming’s product quality assurance that “Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place.” [5].

B. NSE

Software quality assurance is based on defect prevention and defect propagation prevention performed in the entire software development process from requirement development down to maintenance through program execution using the innovated Transparent-box testing method dynamically, plus inspection using traceable documents and traceable source code automatically diagrammed.

2.8 Software Documentation Paradigm

A. The old-established software engineering paradigm

Software documents are separated from the source code without bi-directional traceability between them, and often inconsistent with source code after code is modified again and again.

B. NSE

Software documents are combined with source code through various traceability, and consistent with source code after code modification through backward traceability to update the related documents directly or updating the corresponding database for re-generating most documents from source code – see Fig. 8.

![The NSE Documentation Paradigm](image)

Fig. 8 NSE documentation paradigm

2.7 Software Quality Assurance Paradigm

A. The old-established software engineering paradigm

2.9 Software Visualization Paradigm
A. The old-established software engineering paradigm

Partially supported in the modeling process using UML and the support tools.

B. NSE

With the NSE process model and the support platforms, the entire software development process is visible from the first step to the maintenance phase using integrative and traceable 3J graphics (J-Chart, J-Diagram, and J-Flow innovated by me) and the corresponding diagramming tools, which generate all charts and diagrams globally and holistically with various kinds of traceabilities to make the software product being developed much easier to understand, test, and maintain - see Fig. 8.

![Fig. 9 An application example of NSE software visualization paradigm](image)

2.10 Software Maintenance Paradigm

A. The old-established software engineering paradigm

Based on linear process models without facilities for various bidirectional traceabilities, or very limited traceability made manually; software maintenance is performed locally and partially with no way to prevent the side-effects for the implementation of requirement changes or code modifications, so that often when a bug is fixed, there is a 20% to 50% chance to introduce a new one to the software product. Often the regression testing is performed by reusing all test cases – it is time consuming and costly. It is why software maintenance takes more than 75% of the total cost and total effort in a software system development.

B. NSE

Based on the NSE nonlinear process model with the support of facilities for various bidirectional traceabilities that are automatically established, software maintenance is performed globally and holistically with side-effect prevention. There is no big difference between the software development process and the maintenance process, because with NSE requirement changes are welcome at any time to support the customer’s market competition strategy, and responded to in real time where the side-effects for the implementation of requirement changes or code modifications can be prevented to assure the quality through various bidirectional traceabilities. The regression testing after code modification can be performed with minimized test cases to greatly save the cost and time. In the case that only a few code branches are modified, only some related test cases will be selected for regression testing through backward tracing from the modified branches to the test case scripts. The regression testing will use the Transparent-box method which combines functional testing and structural testing together seamlessly with the capability to establish the new bidirectional traceabilities, and the capability to perform performance measurement, memory leak and usage violation check, and MC/DC (Modified condition/Decision Coverage) test coverage measurement. If something wrong is found after the code modification, a global and holistic version comparison will be performed for helping users to find and fix the problem quickly.

2.11 Software Project Management

A. The old-established software engineering paradigm

The project management processes are separated from the product development processes – the project plan/schedule information and the cost information are not traceable with the requirement implementation, so that often a software becomes a monster of missed schedules and blown budgets.

B. NSE

The project management processes and the product development processes are combined together, making the project plan/schedule information and the cost information traceable with the requirement implementation and the source code, assigning implementation priorities to the requirements according to the importance and market needs, so that the schedules and budgets can be controlled better. Particularly, the NSE nonlinear process model is used with defect prevention for the implementation of requirement changes or code modification to greatly reduce the cost spent in the software development process and the software maintenance process, and ensure the quality from the first step to the end of a software development project.

An application example is shown in Fig. 10.
3. The Essential Differences between the Old-Established Software Engineering Paradigm and NSE

The Essential Differences between the Old-Established Software Engineering Paradigm and NSE is how to handle the relationship between the whole of a nonlinear system and its parts – the old-established software engineering paradigm is based on reductionism and the superposition principle that the whole of a nonlinear system is the sum of its components, so that with it almost all software engineering activities are performed linearly, partially, and locally; but NSE is based on complexity science by complying with the essential principles of complexity science, particularly the Nonlinearity principle and the Holism principle that the whole of a nonlinear system is greater than the sum of its parts, the characteristics and behaviors of the whole of a nonlinear system emerge from the iteration of its parts, cannot be inferred simply from the behavior of its individual components, so that with NSE almost all software engineering activities are performed non-linearly, holistically, and globally.

4. Conclusion

Why have the critical issues (low quality and productivity, and high cost and risk) existed for more than 40 years? The main reason is that software is a nonlinear system where a small change may bring big impact to the entire system. Each one of the critical issue, such as the low quality issue, is related to the all parts of software engineering, including the software development method, the software engineering modeling approach, the software testing paradigm, the documentation paradigm, the visualization paradigm, and the maintenance paradigm, so that only improve the quality assurance method and tools without improving the other parts of an entire software engineering paradigm will not be able to efficiently solve the software quality issue – for efficiently solving the all critical issues in software development, we need a complete revolution in software engineering based on complexity science as described in this paper.

References