Software Development Methodology Revolution Based on Complexity Science - An Introduction to NSE Software Development Method

Chi-Hung Kao¹, Jay Xiong²
¹The Jumpulse Center of Research and Incubation of Northwestern Polytechnic University
²NSEsoftware., LLC., USA

Abstract - This article introduces NSE (Nonlinear Software Engineering) methodology based on complexity science. Complying with the essential principles of complexity science, especially the Nonlinearity and the Generative Holism principles results that the whole of a complicated system may exist before building up its components. The characteristics and behaviors of the whole system emerge from the interactions of its components, so that NSE software development methodology suggests almost all software development tasks and activities are performed holistically and globally. Complying with W. Edwards Deming’s product quality principle 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.”, NSE software development methodology is driven by defect prevention and traceability from the first to the final step in order to ensure the quality of a software product. NSE software development methodology supports top-down plus bottom-up software engineering, makes software design become pre-coding, and coding become further design.

Keywords: software traceability, requirement traceability, validation, verification, testing, quality assurance, maintenance

1 Introduction - Almost All Existing Software Development Methodologies Are Outdated

Almost all existing software development methodologies are outdated because

(1) they are based on reductionism and superposition principle that the whole of a nonlinear system is the sum of its parts, so that almost all software development tasks and activities are performed linearly, partially, and locally.

(2) they are complied with the Constructive Holism principle that software components are developed first, then, as stated in CMMI, “Assemble the product from the product components, ensure the product, as integrated, functions properly and deliver the product.” [1]

2. Outline of the Revolutionary Solution Offered by NSE

The revolutionary solution offered by NSE in software development methodology will be described in detail in this article later. Here is the outline of the solution:

(1) It 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 complex system is greater than the sum of its components – the characteristics and behaviors of the whole emerge from the interaction of its components, so that with NSE almost all tasks and activities in software development are performed holistically and globally. For instance, requirement changes are welcome to enhance customers’ market competitive power, and implemented holistically and globally with side-effect prevention through various traceability to avoid “Butterfly Effects”.

(2) It complies with the Generative Holism principle of complexity science that the whole of a complex system exists (as an embryo) earlier than its components, then grows up with them. As pointed by Frederick P. Books Jr. that “Incremental development – grow, not build software … the system should first be made to run, even though it does nothing useful except call the proper set of dummy subprograms. Then, bit by bit it is fleshed out, with the subprograms in turn being developed into actions or calls to empty stubs in the level below.” [2] “An Incremental-Build Model Is Better – Progressive Refined … we should build the basic polling loop of a real–time system, with subroutine calls (stubs) for all the functions, but only null subroutines. Compile it; test it. … After every function works at a primitive level, we refine or rewrite first one module and then another, incrementally growing the system. Sometimes, to be sure, we have to change the original driving loop, and or even its module interface. Since we have a working system at all times.

(3) We can begin user testing very early, and we can adopt a build-to-budget strategy that protects absolutely against schedule or budget overrun (at the cost of possible functional shortfall).” [3] From the point of view of quality assurance, the NSE software development
methodology is driven by defect-prevention, defect propagation prevention, and traceability that a software quality is ensured from the first step down to the final one supported by various automated and self-maintainable traceability and software visualization. With NSE software development methodology software testing is performed dynamically in the entire software development lifecycle (including the requirement development phase, the product design phase, the coding phase, the testing phase, and the maintenance phase) using the innovated Transparent-box testing method (see Fig. 1 (B2)) [4] which combining functional testing and structural testing together seamlessly – to each test case, it not only checks if the output (if any, can be none – having a real output is no longer a condition to use this software testing method dynamically) is the same as what is expected, but also checks whether the real program execution path covers the expected one specified in J-Flow (a new type control flow diagram innovated by Jay Xiong), and then establishes the automated and self-maintainable traceability among the related documents, the test cases, and the source code to help users finding and removing the inconsistent defects. It means that the NSE software development methodology complies with W. Edwards Deming’s product quality principle 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].

(4) The defect prevention and defect propagation prevention also performed through software visualization in the entire software development process.

Fig. 1 shows a comparison of the software design strategy and the quality assurance strategy between the existing software development methodologies (part A) and the NSE software development methodology (part B).

Part B, the NSE software development methodology:
measurement database for mapping them together, and some keywords written in the test case description part to indicate the types of the related documents, the file locations, and the bookmarks for opening the traced documents from the corresponding locations, so that it can be used to find functional defects, structural defects, and inconsistency defects in the entire software development lifecycle. (B3): The defect prevention is performed mainly through dynamic testing using the Transparent-box testing method in all phases of a software development lifecycle.

The major features of NSE software development methodology:

(1) It is visual – with NSE, the entire software development process and the obtained results are visible, supported by the NSE Software Visualization Paradigm.

(2) The preliminary applications show that compared with the old-established software development methodologies based on reductionism and the superposition principle, it is possible for NSE software development methodology (working with the NSE software development process model) to help software development organizations increase their productivity, lower their cost, improve their product quality tenfold several times, and raise their project success rate.

(3) It brings revolutionary changes to the CBSD (Components-Based Software Development) approach by shifting the software component development foundation base from reductionism and the superposition principle to complexity science in order to greatly ensure the quality of the components themselves, and further make the components adaptive and maintainable as well. According to the principle of complexity science that the behavior and characteristics of a complex system is determined by both the whole of the system and its components, with NSE a software component used for CBSD should at least satisfy the following listed conditions:

* being 100% tested using the MC/DC (Modified Condition/Decision Coverage) test coverage metric, no matter whether it is provided as a class (a class can not be directly executed, so that the test coverage data should be collected through its instances) or a regular function;
* being verified that there is no memory leak or memory usage violation found;
* being verified that it will not become a performance bottleneck to the application system;
* being verified that it will not bring bad effects to the file and the I/O systems for the applications;
* being verified that it satisfies the quality standard in the corresponding applications;
* being verified that it is provided with the related documents, the test cases, and (if possible) the source code traceable from and to the documents.

3 The Driving Forces for the Innovation of the NSE Software Development Methodology

NSE Software Development Methodology is driven by defect-prevention and various automated and self-maintainable traceabilities:

(A) Defect prevention

Repeatable Defect Prevention through:

(a) causal analysis,
(b) preventive actions,
(c) increase awareness of quality issues,
(d) data collection, and
(e) improvement of the Defect Prevention Plan.

New Defect Prevention (more useful) through bidirectional traceability to prevent

(a) inconsistent or changed requirement definitions that may contain conflicts
(b) inconsistent designs or design changes
(c) inconsistent coding (such as inconsistencies between function definitions and calling statements)
(d) inconsistent source code modification, etc.

(B) Traceabilities, including

(a) automated and self-maintainable traceability among documents, test cases and source code, including the documents obtained from project planning, requirement development, product design, coding, testing, and maintenance. This type of traceability is essential to software validation, verification, debugging, and the identification of unimplemented requirements, useless source code modules,
requirements that are related to a module to be changed (for consistent modification), test cases that can be used for regression testing (whereby the efficiency of regression testing can be improved tenfold!), etc. This kind of traceability is established through dynamic testing using the Transparent-box method. Some Time Tags automatically inserted in the test cases description and the test coverage database would build the traceability between test cases and the source code.

(b) the traceability between test cases and the source code has been extended to include all related documents using some keywords written in the description part of a test case to indicate the document formats, the file paths, and the corresponding bookmarks for showing the corresponding locations of the documents.

c) automated and bidirectional traceability within the source code, among source files, classes, functions, and detailed statements is established by diagramming the entire program. For instance, creating the traceability automatically between header file and "#include" statement, program tree and function body, function definition and function call statement, class instance and class definition, goto statement and label, etc. these types of traceabilities are essential for an efficient source code inspection and walkthrough, testing, bug checking, consistent source code modification, etc.

d) capability to trace a runtime error to the execution path and the related functions. This type of traceability is useful for debugging with testing.

e) automated traceability in a systematic analysis of software changes such as version comparison results at the system level, source file level, class and function level, and statement level would be displayed graphically. For instance it includes identifying which modules are deleted (shown in brown), added (shown in green), changed (shown in red), and unchanged (shown in blue). For a changed module, we can further trace the detailed source code to find which statements are deleted, added, and modified. This type of traceability is very useful for version comparison and debugging, particularly in the maintenance phase when some bugs have been removed but new bugs are found.

(f) automated traceability among documents such as those for requirement management as specified in CMMI, including documents for requirement specifications, changes, comprehension, etc., in order to realize this type of automatic traceability, we use a set of predesigned templates in HTML/XML format. These templates will link together by themselves.

(g) automatic traceability through all possible execution paths for each module from a call graph, this kind of traceability is useful in identifying which other modules may be affected due to a change.

4 The related NSE software engineering process model

The NSE software development methodology works seamlessly with the NSE software engineering process model shown in Fig. 2.

NSE software engineering process model consists of three parts – the preprocess, the main process, and the support facility for automated and self-maintainable traceability among the related documents such as the requirement specification, the test cases, and the source code.

The main purpose of the preprocess is to assign priority to the requirements according to the importance, perform prototyping for the important and unfamiliar requirements to reduce risk, execute the function decomposition for functional requirements and system preliminary design through dummy programming to form the whole of a software system. For instance, an embryo using dummy modules contains an empty body including only some calling statements for other low-layered ones without detailed programming logic – see the Completeness Percentage axis of the graphical description of the NSE software development methodology shown in Fig. 3. The “Bone” system (about 5% of the product effort, the first milestone) is obtained in the preprocess.

The implementation of requirements is performed with the main process incrementally through two-way iteration supported by automated and self-maintainable traceability. It is recommended to complete the implementation of about 20% of the most important requirements to form an essential version of the product – see the Completeness Percentage axis of the graphical description of the NSE software development methodology shown in Fig. 3; corresponding to the “Essential” version (second milestone) of the product completeness.

After that, the whole system grows up with more incremental implementations of the requirements, until the final product is completed. With the NSE software development methodology, all versions including the “Bone” system are executable (even if there is no real output provided), and delivered to the customer for review and the feedback may be used for improvement.

As shown in Fig. 2, the NSE software engineering process model is a nonlinear one which assumes that the upper phases might introduce defects or some mistakes, so to check the inconsistency with the upper phases to improve the product is required – as a critical issue is found, there may be a need going back to the preprocess to design a better solution method for the corresponding requirement(s), and perform the prototyping again.
Fig. 2 The NSE Process Model which includes the preprocess part, the main process part, and the automated and self-maintainable facility to support bi-directional traceability using Time Tags automatically inserted into both the test case description part and the corresponding test coverage database for mapping test cases and the tested source code, and some keywords to indicate the related document types such as @WORD@, @HTML@, @PDF@, @BAT@, @EXCEL@ written in the test case description part followed by the file paths and the bookmarks to be used to open the traced documents from the specified positions.

5. Graphical Presentation of the NSE Software Development Methodology

The graphic description of the NSE software development methodology is shown in Fig. 3.

As shown in Fig. 3, there are three axes representing the Work Flow, the Time, and the Completeness Percentage separately.

In the Work Flow axis, it not only includes the phases of requirement development, design, coding, testing, and maintenance, but also includes the project management, the product delivery, and the support for the product web site and BBS for communication – it combines the product development and maintenance together, and furthermore integrates software development and project management together. No matter in what phase, defect prevention and defect propagation prevention is performed to ensure the quality of the product being developed. It does not always follow a linear order – as shown on the right side, upstream movement is supported through traceability for two-way iteration, if necessary.

The Time axis represents the progress. The Completeness Percentage axis shows how many percents of the product are completed – there are three milestones: the first one is the “Bone” system completed through dummy programming; the second one is about 20% of the most important requirements have been implemented; the third one is the final product. The "Bone" version is completed through the preprocess – after prototyping and risk analysis, system decomposition of the functional requirements will be performed, then the decomposition result will be used for the preliminary design to establish the “Bone” system through dummy programming (each dummy module has an empty body or a list of function call statements without detailed program logic). The “Essential” version of the product is completed in the main process incrementally for most important requirements (about 20% of the initial requirements).

Often in the final product, the number of the requirements will be doubled or even more – NSE supports requirement changes in both the software development process, and the software maintenance process with side-effect prevention through various traceabilities.

Additional instructions on sections and subsections

Avoid using too many capital letters. All section headings including the subsection headings should be flushed left.

6 Application
As described above, the NSE software development methodology is driven by defect prevention, defect propagation prevention, and traceability mainly through dynamic testing using the Transparent-box testing method, and software visualization in the entire software development process.

NSE software development methodology supports top-down plus bottom-up approach – design becomes pre-coding, and coding becomes further design as described graphically in Fig. 4.

With NSE, the preliminary design of the whole of a software system is performed in the preprocess (see Fig. 2) through stub programming using dummy modules based on the results of the function decomposition following the functional requirements and the description in the non-functional documentations.

**Directic coding from the design result:**

With NSE software development methodology, coding can be performed directly by editing the dummy modules designed to extent the program logic as shown in Fig. 5.

With NSE software development methodology, coding becomes further design - for instance, in the case that the design shows function A calls function B, but the coding engineers find the function A should call function C and function C should call function B - after coding they can update the design documents by rebuilding the database to make the design result consistent with the code (in this case, they may choose the way to modify the design first, then edit and change the code) as shown in Fig. 6 and Fig. 7.

**Holistic development:**

With NSE software development methodology, the whole of a software system will be designed first using stub programs using dummy modules as shown in Fig. 8.
Static traceability for defect removal through review:

With NSE software development methodology, the design results are traceable for static defect removal as shown in Fig. 9.

System growing up incrementally with defect prevention:

With NSE software development methodology, a software system being developed will grow up incrementally as shown in Fig. 10 and Fig. 11.

Coding style independent visualization support:

With NSE software development methodology, it is supported by a coding-style-independent graphical representation technique and tools, so that the source code written by others is also easy to read and understand – see Fig. 12;

Dynamic traceability among documents and test cases and source code for defect removal:

With NSE software development methodology, quality is ensured mainly through defect prevention and defect propagation prevention based on dynamic testing using the innovated transparent-box testing method to establish bi-directional traceability as shown from Fig. 13 to Fig. 16.
Fig. 13 Tracing a test case to find two inconsistency defects:
(1) The real execution path did not cover the expected execution path
   \[\text{main (int, char**)} \{s0, s3\}\] – segment s3 is highlighted as untested;
(2) The bookmark for opening the global routing section of the prototyping document, \text{g\_router}, pointed to the wrong section – the global placement section.

Removing the defects:
(1) find the location for the first defect and modify the main() program:
From:
\[
\text{if (strcmp(argv[1],"global\_router")\equiv0)}
\quad:\text{calling g\_router(argv[1])}
\]
To:
\[
\text{if (strcmp(argv[1],"global\_router")\equiv0)}
\quad:\text{calling g\_router(argv[1])}
\]
(2) find the mistake related to the bookmark, \text{g\_router} (Fig. 14), and fix it (Fig. 15):

Fig. 15 Fixing the bookmark mistake

7 Conclusion

This article presents the NSE software development methodology based on complexity science. It is driven by traceability, defect prevention, and defect propagation prevention through dynamic testing using the Transparent-box testing method and software visualization. Preliminary applications show that compared with the existing software development methodologies it is possible for the NSE software development methodology (with NSE process model and the support platform) to help software development organizations efficiently solve many critical issues in software development to ensure software quality and increase software development productivity.

8 References


