Abstract
This paper introduces Nonlinear Software Engineering Documentation Paradigm based on complexity science, which brings revolutionary changes to software documentation by making software documents automatable, graphical, colorful, traceable, virtual, maintainable, always consistent with the source code, and dynamic ("alive") – the generated documents are three in one: when a graphical document is shown, the document generator is always working and waiting for users’ commands from the interface (using the document itself) for better software understanding, testing, and maintenance.

Keywords: software documentation, software document, visualization, diagram, testing, maintenance

1. Introduction
As pointed by O PaulI, eHow Contributor that software documentation pervades the software life cycle. It is the visible part of the software process. Without it, software cannot be maintained. Without it, users cannot train and they virtually cannot use the software. Without it new developers would have to re-invent the wheel in software development. Software documentation is the most important manifestation of software. It is the guide through the software maze (http://www.ehow.com/about_6706857_importance-software-documentation.html).

Experience has shown that technical software documentation are unsatisfactorily maintained. The implications of outdated documentation are costly and can even damage the business (http://www.sig.eu/en/Services/DocGen).

Software documents can be created manually, using a "cut and paste" technique, or generated by an automated tool for a large software product.

Unfortunately, often the documents obtained using current software documentation techniques and tools are
* not holistic
* not traceable
* not accurate
* not precise, and
* not consistent with the source code

This paper introduces a new software documentation approach based on complexity science which brings revolutionary changes to software documents to efficiently solve all of the problems listed above.

2. What Does a Revolution in Software Documentation Mean?
According to “The Structure of Scientific Revolutions” [Kuh62], a revolution of software documentation means
(1) Bringing drastic, complete, and fundamental change of software documentation paradigm
(2) Resolving some outstanding and generally recognized problems in software documentation
(3) There is no other way can efficiently resolve those outstanding and generally recognized problems in software documentation.

3. What is NSE Documentation Paradigm?
NSE software documentation Paradigm is a new paradigm which
(1) brings drastic, complete, and fundamental change of software documentation paradigm by
* Shifting the foundation of software documentation from that based on reductionism to that based on complexity science through a paradigm-shift framework FDS innovated as shown in Fig. 1.

Fig. 1 FDS (Five-Dimensional Structure Synthesis method) framework
* Providing complete software documentation support in the entire software development process from requirement development down to software maintenance
* The generated documents are holistic, traceable, virtual, dynamic, accurate, precise, and always consistent with the source code

(2) Resolving some outstanding and generally recognized problems in software documentation

The problems addressed - the Old-Established Software Documentation Paradigm Is Outdated

The Old-Established Software Documentation Paradigm Is Outdated because

(a) The foundation of the old-established software documentation paradigm is reductionism and superposition principle that the whole of a complex system is the sum of its components, so that with the old-established software documentation paradigm almost all tasks and activities in software documentation are performed linearly, partially and locally.
(b) It works with the linear process models in which the workflow goes linearly in one way with one track only without upstream movement at all – it requires the software developers always document the software right without making any mistake and any wrong decision in software documentation, violating the nature law about people because people are linear, easy to make mistakes and wrong decisions to be corrected.
(c) It works with the outdated software development methodologies based on reductionism principle and Constructive Holism principle that the components of a complex system are completed first, then the whole of the system is assembled from its components.
(d) It is not holistic – with it many small pieces of documents will be created/generated without capability to document an entire software system – missing the “Big Picture” of a software product. Even if some tools can be used to document an entire software product, without automated and self-maintainable traceability the obtained system-level graphical documents will still useless, because there will be too many connection lines to make the documents hard to view and hard to understand as shown in Fig. 2.
(e) The graphic documents and the source code are separated - hard to keep them consistent.
(f) The obtained documents are not traceable – hard to use.
(g) Often the obtained documents are inconsistent with the source code after product modification
(h) The documents obtained are stored statically as hard copies in Postscript, XML, or other format, requiring huge amount of space, and long loading time.
(g) Most graphic documents are created manually or using graphic editors, not automatically generated.
(i) The graphic documents are time-consuming to draw, hard to change, and hard to maintain.
(j) Often the obtained documents are not accurate.
(k) Often the obtained documents are not precise – for instance, it can not directly and graphically show where a code branch or condition combination has been tested or not.

**Outlines of NSE Software Documentation Paradigm:**

(a) The foundation of the NSE documentation paradigm is complexity science complying with the essential principles of complexity, particularly the Nonlinearity principle and the holism principle, so that with NSE almost all software documentation tasks and activities are performed holistically and globally.
(b) It works with NSE process model in which the workflow goes nonlinearly through two-way interaction with multiple tracks – supporting upstream movement and downstream movement.
(c) It works with NSE software development methodology based complexity science and the Generative Holism principle that the whole of a complex system is comes first as an embryo, then grows up with its components.
(d) It is holistic – with it many holistic documents for an entire software product will be automatically generated to make the documents easy to view and easy to understand as shown in Fig. 3, and Fig. 4.

**Outlines of NSE Software Documentation Paradigm:**

(2) Resolving some outstanding and generally recognized problems in software documentation

The problems addressed - the Old-Established Software Documentation Paradigm Is Outdated

The Old-Established Software Documentation Paradigm Is Outdated because

(a) The foundation of the old-established software documentation paradigm is reductionism and superposition principle that the whole of a complex system is the sum of its components, so that with the old-established software documentation paradigm almost all tasks and activities in software documentation are performed linearly, partially and locally.
(b) It works with the linear process models in which the workflow goes linearly in one way with one track only without upstream movement at all – it requires the software developers always document the software right without making any mistake and any wrong decision in software documentation, violating the nature law about people because people are linear, easy to make mistakes and wrong decisions to be corrected.
(c) It works with the outdated software development methodologies based on reductionism principle and Constructive Holism principle that the components of a complex system are completed first, then the whole of the system is assembled from its components.
(d) It is not holistic – with it many small pieces of documents will be created/generated without capability to document an entire software system – missing the “Big Picture” of a software product. Even if some tools can be used to document an entire software product, without automated and self-maintainable traceability the obtained system-level graphical documents will still useless, because there will be too many connection lines to make the documents hard to view and hard to understand as shown in Fig. 2.
(e) The graphic documents and the source code are separated - hard to keep them consistent.
(f) The obtained documents are not traceable – hard to use.
(g) Often the obtained documents are inconsistent with the source code after product modification
(h) The documents obtained are stored statically as hard copies in Postscript, XML, or other format, requiring huge amount of space, and long loading time.
(g) Most graphic documents are created manually or using graphic editors, not automatically generated.
(i) The graphic documents are time-consuming to draw, hard to change, and hard to maintain.
(j) Often the obtained documents are not accurate.
(k) Often the obtained documents are not precise – for instance, it can not directly and graphically show where a code branch or condition combination has been tested or not.
(e) With NSE source code (either a stub program or a regular program) is the source for most graphical document generation, while the graphic documents are the visual face of the corresponding source code — so that they are always consistent each to other.

(f) The obtained documents are traceable to and from the source code — easy to use.

(g) The obtained documents are consistent with the source code after product modification — only needing to update the database.

(h) The documents generated directly from the source code and the corresponding database are virtually existing without storing hard copies in memory or hard disk (unless the users want them) to greatly save the space and make the speed of display about 1000 times faster.

(g) With NSE almost all graphic documents are automatically generated.

(i) The graphic documents are easy to generate, change, and maintain.

(j) The generated graphical documents are accurate to the code.

(k) The generated graphical documents are precise — for instance, it can directly and graphically show where a code branch or condition combination has been tested or not.

(3) There is no other way which can efficiently resolve those outstanding and generally recognized problems in software documentation

Although there are many software documentation tools available on the market, all of them are outcomes of linear thinking, reductionism, and the superposition principle that the whole of a software product is the sum of its parts, so that with them almost all software documentation activities are performed linearly, partially, locally, and qualitatively. For instance, when a source module is to be modified, none of the existing software documentation tools can provide quantitative information about how many requirements may be affected and how many function call statements may need to be modified too without various bidirectional traceability.

4. Description of the NSE Documentation Paradigm

NSE documentation paradigm generates holistic, graphical, interactive, and traceable documents automatically from the source code of a stub program for requirement development and product design or a regular program. In forward engineering or reverse engineering.

As shown in Fig. 5, with NSE the source code (either a dummy program or a regular program) is also the source for automatically generating the graphic design documents, while the graphical design documents become the visual faces of the source code — design becomes pre-coding, and coding becomes further design. The designed graphical documents are traceable for static review as shown in Fig. 4 and executable for dynamic defect removal.

With NSE there is an automated and self-maintainable facility established through dynamic testing using Transparent-box method to support traceability among the design documents and the test cases and the source code as shown in Fig. 5 — it makes the design documents and the source code consistent. After the source code is changed with the implementation of requirement changes and code modifications, the most design documents can be automatically and incrementally (only the modified source files needed to be re-analyzed) updated to keep consistency with the source code.

Although

The NSE Documentation Paradigm

The Objectives of the NSE Documentation Paradigm

The objectives of NSE documentation paradigm are:

(a) combining software programming and graphical software documentation together seamlessly;

(b) making one source for both people understanding and the computer “understanding” - through static people review of the graphical documents and dynamic program execution to ensure the upstream quality of a software product.
(c) realizing all kinds of documents (including those manually drawn and those generated by third parties’ tools) traceable to source code to keep them consistent each to other through automated and self-maintainable traceability established by dynamic testing using Transparent-box method combining functional testing and structural testing together seamlessly with capability to establish bidirectional traceability.

(d) generating most software documents automatically as more as possible.

(e) Making software documents visible as more as possible.

(f) With the graphical documents consistent to the source code, making a software product truly maintainable and adaptive to the changed or changing environment.

Working with stub Programming

NSE software documentation paradigm works with dummy programming using dummy modules consisting of an empty body or only simple function call statements - any software professional can write the dummy programs easily without extra learning.

Working with NSE Software Visualization Paradigm

NSE documentation paradigm works closely with NSE Software Visualization Paradigm which mainly generates interactive and traceable 3J graphics (J-Chart, J-Diagram, and J-Flow diagram). As described above, making software documents visible as more as possible is one of the objectives of NSE software documentation paradigm.

With HAETVE, classes are represented in several graphical notations as shown in Fig. 6.

(d) Time-Event table

With NSE, the time-event table is written in the comment part of a stub program or a regular program. An example is listed as follows:

(e) Mapping to Activity diagram

With NSE, a new type logic diagram – J-Diagram is used to map to Activity diagram.

(f) Method for graphically representing message sending and receiving

With NSE, message sending and receiving are represented with the automatically established “click-to-jump” facility as shown in Fig. 7.

How it works

Fig. 8 shows the workflow for NSE software documentation paradigm.

Making a software product visible in multiple-views

1. Static + dynamic

(a) Static analysis of a program Cyclomatic complexity – see Fig. 9.
(b) Dynamic analysis of a program performance - see Fig. 10.

(2) Macro + micro
(a) Holistic MC/DC test coverage analysis for an entire software product – see Fig. 11.

(b) Detailed MC/DC test coverage analysis for a individual class/function – see Fig. 12.

(3) Procedure + data

(a) Function cross-reference analysis – see Fig. 13.

(b) Data analysis – see Fig. 19-18.

(4) System-level + file-level + statement-level
(a) System-level version comparison – see Fig. 14.

Fig. 14 AN application example of holistic version comparison in system-level

(b) File-level version comparison – see Fig. 15.

Fig. 15 An application example of file-level version comparison

(c) Statement version comparison – see Fig. 16.

Fig. 16 AN application example of statement version comparison

(5) Static visibility + dynamic visibility

(a) Forward tracing from a test case to find what modules can be tested – see Fig. 17

Fig. 17 Example of static visibility – tracing a test case to view what modules can be tested

(b) Dynamic visibility – tracing a test case to not only find what modules can be tested, but also directly pay the captured test operations back through the batch file (.bat) specified in the @BAT@ keyword within the test case description part – see Fig. 18.

Fig. 18 Dynamic visibility – tracing a test case to play the captured operations back

(6) Integrative + traceable

(a) With NSE the generated documents are interactive – for instance, user can click on a module-box to use that module as the root to generated a sub-call-graph – see Fig. 19.
Fig. 19 Interaction example: click on a module-box to generate an isolated sub-call-graph
(b) With NSE most of the generated documents are traceable - see Fig. 20.

Fig. 20 An application example – tracing a module to see the all related modules

(7) Linkable + convertible
(a) With NSE, different graphical documents can be linked together – see Fig. 21.

(b) Converting between the generated logic diagram and the control flow diagram – see Fig. 22.

Fig. 21 An application example - linking a call graph to the logic diagram

Fig. 22 An application example of diagram conversion from a logic diagram to the control flow diagram
(8) Local + internet
(a) With NSE, many static and dynamic analysis reports can be automatically generated – see Fig. 23.

(b) With NSE the generated reports for static and dynamic program analysis can be saved in HTML format to be used as web pages – see Fig. 24.

Fig. 23 An application example of static and dynamic program analysis and reporting

Fig. 24 Code analysis reports saved in HTML format to be used as web pages
The Major Features of NSE Documentation Paradigm

The graphical documents generated by NSE documentation paradigm are:

- **Holistic** – NSE documentation paradigm generates holistic charts and diagrams to document an entire software product.
- **Interactive** – the generated graphical documents are interactive, the generated charts/diagrams themselves are also the interfaces to accept user’s commands/operations.
- **Traceable** – with NSE most of the generated documents are traceable, useful for validation, verification, and semi-automated inspection and walkthrough.
- **Accurate** – with NSE the source code of a dummy program or a regular program is also the source to automatically generate most graphical documents, so that the generated documents are accurate and consistent to the source code.
- **Precise** – the generated graphical documents are precise, for instance, the corresponding documents can show how many times a branch is executed, and what code branches and conditions have not been executed.
- **Virtual** – with NSE the most graphical documents are dynamically generated from source code, there is no need to save their hard copies in memory or disk, so that a huge amount of space can be saved, and the display speed is about 1000 times faster. The generated holistic charts and diagrams are shown within a Window no more or less. When a chart or diagram is needed to move, a new one will be regenerated dynamically without real moving the chart or diagram, so that the display speed is very fast, but form a users’ point of view, there is no difference between the virtual charts/diagrams and the regular charts/diagram needing huge amount of space to store in hard disk and the computer memory.
- **Massive** – the graphical documents with the size more than 100 times of the size the source code (if the documents are stored in disk regularly) can be automatically generated in system-level, file-level, and module-level. For instance, for each class/function, NSE documentation paradigm can automatically generate the logic diagram shown in J-Diagram notations with the untested branches and untested conditions highlighted, control flow diagram shown in J-Flow diagram notations, the quality measurement result shown in Kiviat diagram, etc. – massive.

Application

NSE documentation paradigm has been commercially implemented and supported by Panorama++. The all screenshots shown in this chapter come from real application examples.

Summary

The old-established software documentation paradigm is outdated because it is based on reductionism and superposition principle that the whole is the sum of its components, so that with it almost all software documentation tasks and activities are performed linearly, partially, and locally. The sources used to generate/create software documents are different from the source code of the software product. The generated/created graphical documents are not traceable for static review, and not executable for dynamic testing, so that the quality of the documents is hard to ensure, and the documents are hard to keep consistency with the source code after code modifications.

NSE software documentation paradigm 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 its components, and that the characteristics and behaviors of the whole emerge from the interaction of its components, so that with NSE software documentation paradigm almost all software documentation tasks and activities are performed holistically and globally. The sources used to generate most graphical software documents are also the source code of the dummy programs or regular programs. The generated graphical documents are traceable for static review, and the corresponding source code is executable for dynamic testing, so that the quality of the documents is easy to ensure, and the documents are easy to keep consistency with the source code after code modifications – with NSE design is pre-coding, while coding is further design.

Source code is not the best documents of a software product, but source code is the best source to directly and automatically generate holistic, interactive, traceable, consistent, accurate, precise, massive, and virtual documents of the software product.

19.7 References