The DP&T Model: The *Defect Prevention and Traceability*–Driven Model for Software Engineering

Jay Xiong  
International Software Automation, Inc.,  
& ISA Shanghai, Ltd.,  
Shanghai, China,  
www.ISApanorama.com, jay@ISApanorama.com

Jonathan Xiong  
Electrical Engineering & Computer Science,  
University of California, Berkeley,  
2650 Durant Ave., Freeborn #801,  
Berkeley, CA 94720, USA, jxiong@berkeley.edu

**Abstract:** This paper describes a new type of model for software engineering: the DP&T Model. It is an important part of the new DP&T System that is driven by defect prevention and traceability. Differing from traditional linear models, the DP&T Model is nonlinear, with the capability for forward and backward traceability. This model is used to develop new software products, prevent defects, and fix inconsistencies among all artifacts generated in every phase of the software development life cycle.

**Keywords:** model, defect prevention, traceability, software engineering

1 The DP&T System and DP&T Model

The Defect Prevention and Traceability (DP&T) System is a new type of system for software engineering that is driven by defect prevention and traceability. It consists of six parts: the DP&T Model, the DP&T Methodology, the DP&T Support Graphics, the DP&T Technologies, the DP&T Support Tools, and the DP&T Support Platforms.

The DP&T Model is a new type of model for software engineering. It differs from traditional linear models as shown in Fig. 1 because it is nonlinear and is capable of forward and backward traceability.

---

*Fig. 1 – Traditional linear models.*
This traceability is used to prevent defects and fix inconsistencies among all artifacts generated in every phase of the software development cycle: requirement analysis, design, coding, testing, and maintenance. With the DP&T Model, it is possible to trace from any phase to any other phase.

2 Graphical description of the DP&T Model

The DP&T Model consists of two parts – the main process (shown in Fig. 2) and the additional process (shown in Fig. 3).

Fig. 2 – The main process of the DP&T Model.

Using the DP&T Model is recommended for incremental software product development. Each time, only one or a few subrequirements should be handled through the main process and integrated into the previous working version (if it exists) to form a new working version of the product, even if it is still incomplete.

The additional process is used for prototyping and risk analysis of critical subrequirements.
Fig. 3 – The additional process of the DP&T Model.

Fig. 4 – An example of fixing a bug with the DP&T Model through forward and backward traceability to identify and modify corresponding documents and code.

Figure 4 shows the processes for fixing a bug found in the product design after delivery.
3 The capability for forward and backward traceability

It is easy to understand that bidirectional traceability among all artifacts generated in every phase of the software development life cycle is essential for effective software validation and verification, simple debugging, consistent code modification, and efficient regression testing.

As described in CMMI (Capability Maturity Model Integration) [1], the level 2 REQM (Requirement Management) includes the following:

*SP1.5-1: Identify Inconsistencies Between Project Work and Requirements* and
*SP1.4-2: Maintain Bidirectional Traceability of Requirements.*

But the problem is, “How can these be satisfied?”

Often, a static *traceability matrix* is manually built and used in most cases. However, there are many problems with a static traceability matrix:

- It takes too much time to build a forward traceability matrix manually.
- It is almost impossible to build a backward traceability matrix manually (considering that there can be 10,000 functions in certain programs).
- A static traceability matrix is defective in many cases; errors may come from faulty design or from mistakes made by the engineers building the static traceability matrix.
- A static traceability matrix can only be built at the module level, because a static traceability matrix at the code branch level takes too much time and costs too much.
- A static traceability matrix is hard to maintain after code modification, such as in the case where a module is subdivided into three smaller modules.

What we really need is a bidirectional traceability facility that is dynamically and automatically built during the execution of the developing product using test cases. It should be detailed enough down to the code branch level. It should be updated automatically after code modification, etc.

Now we have the solution. The key points of the facility developed by us for bidirectional traceability include the following:

- based on dynamic execution of test cases specifically designed for the program under development,
- built with correspondence analysis between the test cases and the code,
- implemented using mapping operations according to the corresponding time tags that are automatically added into test script files after program execution,
- precise and accurate all the way down to the code branch level,
- updated automatically after code modification,
- based on using color graphics for all tracing operations, and
- designed with the capability to automatically open related documentation or files when a certain section is traced.

Figure 5 shows how a time tag is added into a test-case script file. An application example of using this facility for forward traceability is given in Fig. 6, provided by Panorama®++ for C/C++, which was developed by International Software Automation, Inc. (www.ISapanorama.com).

In Fig. 6, the test case selected is automatically shown in blue on the test script file. The corresponding files, classes, and functions traced are shown in red on the program control flow diagram, where untested modules are indicated with small black boxes. At the same time, the related requirement specification and design document are opened automatically.
An application example of backward traceability is shown in Fig. 7. In the case in which a user wishes to change the requirement of the ADDITION CALCULATION, we find through forward traceability that the function of symbol_table_add_value() should be modified. But that function may be used for the
implementation of more than one requirement, so to prevent defects from requirement conflicts we use backward traceability to check that function. The result shows that the function is related to the implementation of both the ADDITION CALCULATION and the SUBTRACTION CALCULATION, so modifying the function symbol_table_add_value() should be done carefully to fulfill both requirements.

Figure 8 shows what happens if a certain code segment shown in a control flow diagram (where the untested branches are indicated with small black boxes) is changed. In this example, we find that only one test case is needed to retest the code segment if it is modified. Regression testing becomes much more efficient, since we only need to retest a single test case.

Fig. 7 – An application example of backward traceability at the module level.
4 The benefits of using the DP&T Model

The benefits of using the DP&T Model include making
- requirement validation and verification much easier to perform by utilizing forward traceability,
- inconsistencies among all artifacts generated in every phase of the product development life cycle easier to find and solve,
- defect prevention easier to perform,
- software maintenance much easier to perform, and
- regression testing after code modification much more efficient.

5 Conclusion

The new DP&T Model based on forward and backward traceability is presented in this paper. The DP&T Model can be used to build a new software product efficiently. It is an important part of the DP&T System for software engineering. The DP&T Model is highly useful for requirement validation [2], defect prevention, consistent code modification, and efficient regression testing after code modification.

6 References