The Synergy Approach™

Raghav S. Nandyal
Chief Executive Officer

SITARA Technologies Pvt. Ltd.
'Bellevue', #559 • Road No. 3 • Banjara Hills • Hyderabad AP - 500 034 • INDIA
Email: raghav_nandyal@SITARATECH.com
URL: http://www.SITARATECH.com

A Near Zero-Defect Approach to Software Development

From The

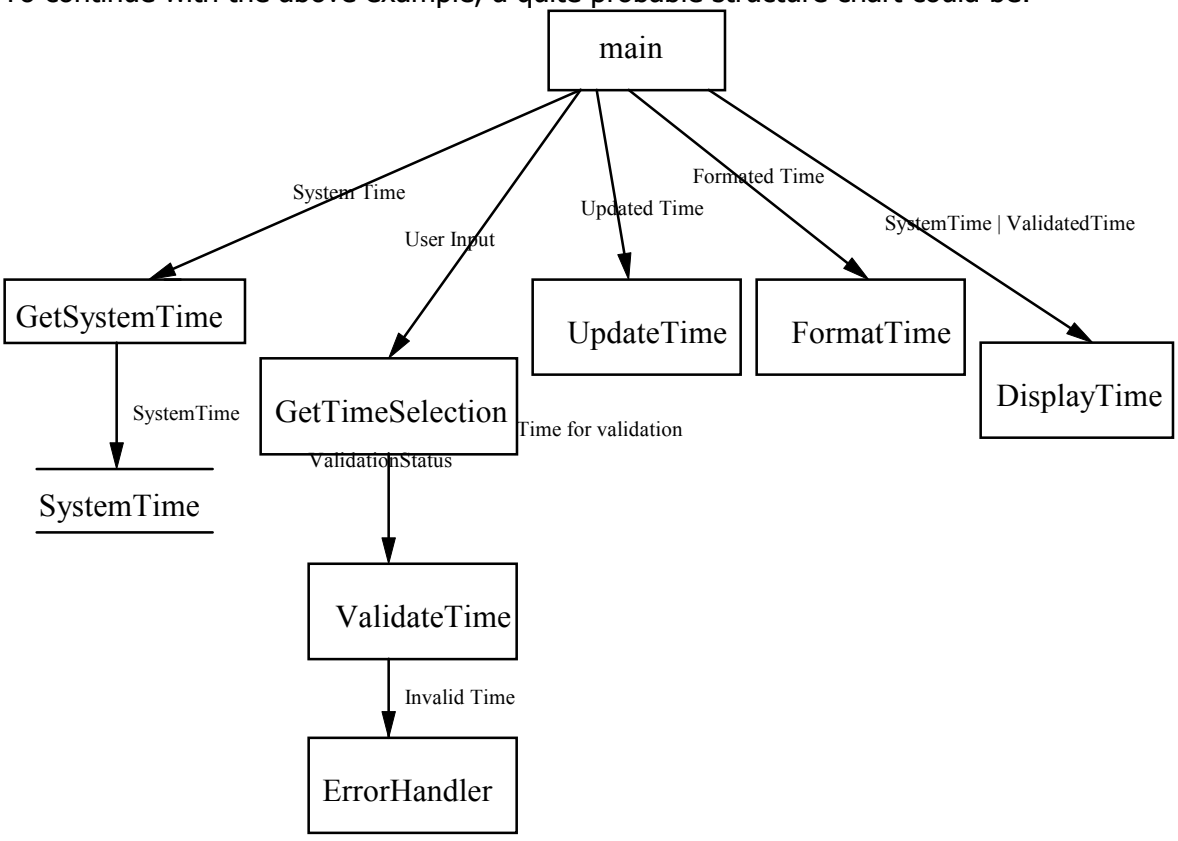
Software Engineering Laboratory

March 1995

2.3 Detailed Design Phase

In this phase, the processes identified in the previous phase are grouped into logically related modules. These modules will then be arranged in a logical sequence from left to right forming the sequence in which they shall be called. Many approaches to design exist today and all of them merit a mention. However, in order to restrict the concerns to good "software engineering practices", this document takes the "structured methods" approach to design. The strategies available for detailed design and how to do it will form a separate document.

To continue with the above example, a quite probable structure chart could be:



The module specification (MSpec) will detail more elaborately the functionality that will be accomplished. It will have the logic detailed in as elaborate a fashion as possible. Ideally, the translation from the MSpec to the code should be a 1:1 ratio. The MSpec for ValidateTime could look like:

```
ValidateTime(Time_for_Validation) RETURNS either VALID_TIME or INVALID_TIME
BEGIN
        IF time format is in 24 Hours format
        THEN
                Validate time against 24 Hours 0 Minutes.
        ELSE
                Validate time against 12 Hours 0 Minutes.
        ENDIF.

        IF time is VALID
        THEN
                return VALID_TIME
        ELSE
                CALL  ErrorHandler() with severity value.
                return INVALID_TIME
        ENDIF
END
```

The end of phase deliverables are:

1. Module Specification (MSpec)
2. Structure Charts
3. Interface Design Description
4. Data Dictionary entries which can become header files.

These form the inputs to the coding phase.

2.4 Coding or Implementation Phase

Once the detailed design is baselined, the implementation is rendered in the language of preference. Among other considerations such as maintainability, portability issues and code structuring is the most important consideration in this phase. Besides of course, code readability, the comment to non comment ratio and ease of understandability (non cryptic coding) will form quality check points during code walk-through and reviews.

One of the most important inputs to this phase is the coding standards document. Every language has its features and is fairly "style" independent. If the code walk-through and reviews must happen in a coherent fashion, it is very important to have a guiding principle in the form of a coding standards document which will suggest best practices in the industry. The coding standards document could be as elaborate as to have "variable naming conventions" (Hungarian notation of MS Windows programming for instance). The coding standards must be documented in a way to help the automation of the code standardization phase. Besides coding, it is important to specify any additional requirements that the code must satisfy, such as passing "lint". The coding standards must also detail the code header format to bring uniformity to the way the code is structured. Appendix D1.1 contains the C coding standards developed by the author.

The deliverable at the end of this phase is the code unit that might require code standardization and unit testing.

2.5 Code Standardization and Unit Testing Phase

Code standardization is necessary to bring uniformity to the way code is developed in the organization. Among other things, code standardization will involve formatting of code. Many tools are available in the UNIX environment such as the C beautifier (cb) and indent.

Unit Testing could be either a black box activity emphasizing functionality or a white box approach emphasizing structure. A better approach is to have a mix of both these approaches so that a disciplined code structure, emphasizing the end-user's perspective could be realized.

In order to keep the process and the organization a learning and continuously improving entity, it is very appropriate to suggest that we use what may be termed a "**Red Book**" and a "**White Book**". The Red Book will besides having a red wrapper, contain "**documentation of the pit-falls encountered during the phase of Unit Testing**" and the "White Book" **containing "lessons learnt from the earlier phases"**. It is in this phase that most of the project understanding reaches a very stable maturity and also forgotten because nobody bothered to document the lessons that were learnt and things that could have been done differently but were not done. From experience Unit Testing and System Integration/Testing is a very useful phase to learn from mistakes. So, a well-documented Red Book will tremendously aid in the next phase, namely System Integration and System Testing phase. A key input to the organizational metric collection effort from these two phases is statistical measures for:

1. Computational errors
2. Logic errors
3. Input and output errors
4. Data handling errors
5. Interface errors
6. Data definition and database errors

The deliverables at the end of this phase are:

1. Debugged code units.
2. Unit Testing Report.
3. Unit Test Cases.
4. **Red Book**
5. **White Book**

2.6 System Integration and System Testing Phase

System Integration can now proceed with debugged code units as inputs in a truly bottom-up fashion. This process of system integration can be perceived as being made up of many smaller integration steps. Finally, after the system is built in such an incremental fashion, it is subjected to a system test. Ideally, the input to the system test phase is the User's manual for the system. This will then bring out the end-user's perspective to testing.

In systems where bug fixes begin to happen in the System Integration phase, it is very likely that overall system behavior would have to be stressed over and over again to gain confidence in the solution. Here, regression tests would have to be conducted and the regression test reports must be well documented. This is a very good area from the point of view of software automation.

It is important to update the Red Book and White Book.

The deliverable at the end of this phase is :

1. System Testing Report.
2. System Test Cases.
3. Traceability of system test cases with the original requirements.
4. Update Red and White Books.
5. Regression Test Report.

2.7 α β and Customer Approval

The basis for this phase will be the System Test Reports and Test Cases along with the Requirements Book and the User's Guide.

3.0 Continuous Process Improvement

The most important inputs to this phase are the Red Book and the White Book.

One of the key points for quality management is to improve constantly. This means, never to keep the software production process static but to improve quality and productivity constantly. Software development methodology, standards and practices must be constantly revised to accommodate newer technologies. An organizational process improvement group (OPIG) must be constituted with a strong management presence to revise and expand the scope of software production, all the while emphasizing "**production of zero defect products with reduced cycle time and increased productivity**". It may not be an overemphasis to state again, the importance of automation and for the process to be a learning entity. From experience, it is possible to automate as much of the more formal phases of the software production process and every effort towards making this possible must be made. *This will hold the key to make the Zero Defect Production Process - The Synergy Approach a truly optimizing process*.

4.0 Conclusion

An ideal software production process will have in it the virtues of being an "optimizing" process. This will ensure that the end of phase deliverables are complete, correct and consistent. And, in order to make the process of developing software repeatable with consistent results, automation of the software production process needs to be addressed as soon as possible. One very important point which must never be forgotten is that - "the customer is NOT buying the process, but he is buying the product". It is for the customer that the software is being developed, so without the customer, it is meaningless. In this context, "process is only a means to an end but NOT an end in

itself". Oftentimes, emphasis on process is so fanatical that, the true spirit of following a process namely, "production of zero defect software", is often forgotten. In fact, there have been situations wherein process adherence has been viewed with much bitterness by both the developers and the customer, especially when doing so has caused a slippage in delivery. It may NOT be prudent to blame the process for poor planning and failure to tailor the process. What appears in this document must be considered in its spirit. Process must bolster the very end objective - zero defect software by imposing a discipline to the much needed task of developing software. The process must add value to the way software is developed and value addition must come in the form of automating the transitions across the more formal phases. A mature software production process must be predictable and provide for-

- Good configuration management or version control mechanisms
- Metric collection
- Reuse library archival
- Automated solutions to the production process
- Defect containment by effective reviews and quality audits
- Project estimation based on data collection

5.0 Closing Remarks

Branches of the Structured Analysis Family

Everything that goes on in a system can be classified into three categories:

1. *What is being done:* the process, operation or algorithm.
2. *To what it's being done:* the data, material or information content of the system.
3. *When it's being done:* this is commonly called control.

Functional Decomposition: The Process Oriented Branch

This was introduced by Ed Yourdon, et al. For real time analysis the work is primarily due to Hatley and Pirbhai.

The analyst starts out thinking of the system (meaning the problem being analyzed) as being a single large process. He then divides that primary process into two or more sub-processes and describes the data connections between them. This activity is repeated until a large number of simple, primitive processes have been defined.

The next step centers on control. The order of execution of the processes is then defined through state models.

Organization of data comes about as a derivative of the processes and their need to link together. Relatively little is said on this matter in the methods based on functional decomposition.

The characteristics of this branch are: PROCESS, CONTROL, DATA

Advantages:

- The most widely known method.
- Intuitively appealing and therefore easy to comprehend.
- In the non real time form, the model notation is simple.

Disadvantages:

- Drives the implementation toward more code and less data.
- The method is non-reproducible, in that two analysts working the same problem often produce very different models.

Event-Response: The Control Oriented Branch

This method first appeared in Essential System Analysis, by Stephen McMenamin and John Palmer in 1984. The analyst views the system as a black box that makes pre-planned responses to external stimuli. The analyst first identifies discrete incidents that occur outside the system, known as "external event". These external events can be readily derived from requirements; they act to describe the system boundary precisely.

Members of this branch of SA look like: CONTROL, PROCESS, DATA

Advantages:

- Fairly reproducible.
- The models produced by this method are easy to understand.

Disadvantages:

- Assumes requirements are complete and consistent; highly unstable if requirements change.
- Leads to code-heavy implementation.
- The real-time extensions use a complex notation.

Object-Oriented Analysis: The Data Oriented Branch

This was first developed in 1979 but appeared as a published material by Shlaer, Mellor in 1988. This approach starts by defining objects and attributes- that is, the data- derived from the real world problem. Lifecycles of the objects are then formalized in state models; this is based on operating policies and physics of the world.

The hallmark of OO branch is: DATA, CONTROL, PROCESS

Advantages:

- Reproducible.
- Drives implementation toward minimal data-driven code.
- Stable when requirements are changing.
- Simple, elegant notation.

Disadvantages:

- Not always easy to see what happens when the system is presented with an external event.
- Because of the non-traditional "look and feel", skepticism sometimes arises when the method is introduced: it isn't intuitively obvious that the method will work.

END OF LECTURE 2

CONTINUED ON LECTURE 4 [SITARA SE/AUGUST 2001]