the Availability Digest

Document Your System

November 2006

Why Don’t Systems Get Documented?

Perhaps the activity that gets the most lip-service from management is *documentation*. But also, perhaps the activity that is most likely to get cut due to budget and time considerations is *documentation*.

With mission-critical systems, documentation takes on an even more important meaning since failover and recovery procedures must be well-documented (and practiced). But good documentation goes much further than that.

Think of all the non-maintainable but critical applications that are out there now because documentation was never completed or was not kept up-to-date. Think of those operational failures that occurred because ordinary procedures were not well-documented.

One of the reasons that documentation ends up at the bottom of the priority list is that it is viewed as too costly and too time consuming. In part, this reputation is well-founded because of an over-zealous approach to documentation in the past. Organizations tried to document in far too much detail. The more detailed the documentation, the more difficult it was to keep it maintained. Compounding this problem is that most people just don’t like to document.

With today’s self-documenting programming languages such as Java, the level of required documentation should certainly be revisited. Coupled with modern documentation standards and the powerful tools that are now available to create documentation according to these standards, there is now little excuse for not providing at least the core system documentation necessary to operate and maintain your systems over the long run.

This article is motivated by a documentation solution adopted by a large company as presented recently at a major computer conference. This company is a major supplier of services to the financial exchange community. This paper draws heavily from that presentation.

The company has decided to use the Universal Modeling Language, UML, as its documentation standard. This article explores its use of UML.

The Core Documentation

It does not make sense anymore to provide detailed documentation of code. It is expensive, time consuming and difficult to maintain, especially when the development staff is always running in "crunch mode." Today's programming languages, properly commented, provide easily readable roadmaps through the code.

However, what is not provided is an overall structure that relates how modules interact with each other and how they combine to perform a business function. More importantly, code documentation does not provide the procedures required for operations, development, and support people to properly do their jobs. Without this documentation, people are leaning upon their experience and good judgment to do the job right. This doesn't always work.

In attempting to find a documentation standard, the company faced several challenges:

- Its systems undergo frequent changes to keep up with business needs, new technology, and capacity growth,

- There were culture clashes among the development groups. Each had its own documentation practices.

- Documentation is a time-consuming job. If the bar is too high, it simply will not get done.

The company listed the following needs that it sought to satisfy with a new documentation standard:

- Development
  - requirements
  - functionality
  - architectural design
  - bug fixes
  - tracking known errors

- Operations
  - how to function
  - what to expect
  - recovery from failures
  - who to contact when problems occur
  - training
  - dependencies

- Support
  - installation procedures
  - configuration settings
  - startup
  - shutdown
  - troubleshooting

Enter UML

Years ago, in the dawn of object-oriented technology, when documentation was often being ignored, there were an abundance of attempts to come up with documentation standards for modeling systems. It was felt that if such standards were adopted, tools would follow which would make the documentation task much easier.

The number of identified modeling languages exceeded fifty in the early 1990s and fueled the "method wars." As early as 1985, James Martin's book *Diagramming Techniques for Analysts and Programmers* described in detail almost two dozen different modeling techniques.

In the mid-1990s, some of these methods began to coalesce. In particular, three researchers in the field, each with his passionately supported method, began to work together to incorporate each other's techniques into a common modeling language. They were Grady Booch, Jim Rumbaugh, and Ivar Jacobson. They became known as the "Three Amigos" for their frequent arguments with each other regarding methodological preferences.

Ultimately, their work was accepted by the OMG (Object Management Group) and became known as the Unified Modeling Language (UML).[1] Today, UML is the accepted standard for documentation; and many powerful products exist to help one easily create UML documents.

UML defines many different modeling diagrams that can be grouped into three categories – Structure, Behavior, and Interaction. The diagrams include:

- **Structure Diagrams** model the structure of physical or conceptual things.
    - *Class Diagrams* describe the types of objects in a system and their relationships.
    - *Component Diagrams* show the software components of a system and how they are related to each other.
    - *Composite Structure Diagrams* are similar to component diagrams.
    - *Deployment Diagrams* show the physical relationships between the hardware and software of a system.
    - *Object Diagrams* are used to explain instances of objects with complicated relationships.
    - *Artifact Diagrams* show the physical constituents of a system assigned to a computer.
    - *Package Diagrams* group related classes into packages.
- **Behavior Diagrams** depict the ways in which structures can behave.
    - *Activity Diagrams* describe the states of activities by showing the sequence of activities performed.
    - *State Diagrams* are similar to Activity Diagrams. They describe the behavior of a system by describing all of the possible states of an object as events occur.
    - *Use Case Diagrams* describe the interaction between users and the system.
- **Interaction Diagrams** model the behavior of use cases by describing the way that groups of objects interact to complete a task.

[1] See the review of their book, *The Unified Modeling Language User Guide*, in this month's *Recommended Reading*.

- *Sequence Diagrams* demonstrate the behavior of objects in a use case by describing the objects and the messages they pass.

- *Communication Diagrams* are similar to Sequence Diagrams. They show the relationship between objects and the order of messages passed between them.

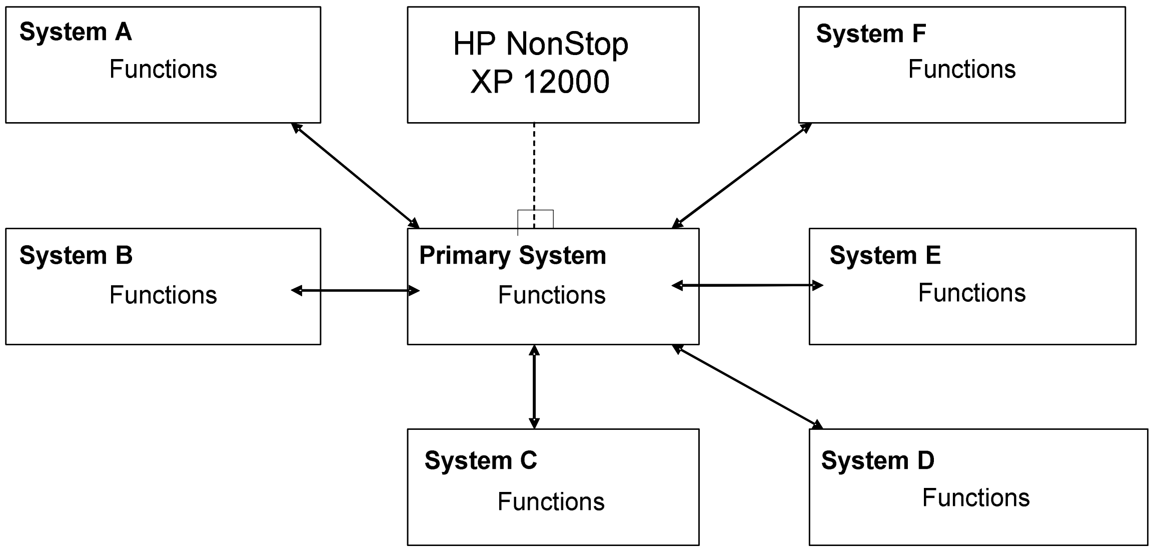
- ***I**nteraction Overview Diagrams* are a form of activity diagram in which the nodes represent interaction diagrams. They introduce two new elements, interaction occurrences and interaction elements.

- *Timing Diagrams* are used to display the change in state or value of one or more elements over time.

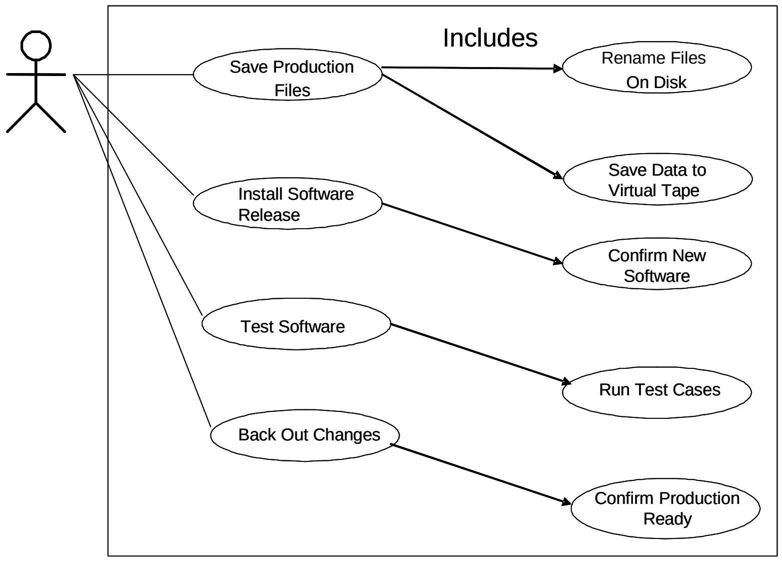
Which Diagrams to Use

The company decided that four of these diagrams satisfied all of their documentation needs. These included the following, and examples from the presentation are included to illustrate the use of each diagram.

- *System Deployment Diagrams* represent the physical packaging of the software modules and their inter-dependencies. The dependencies between components show how changes made to one component may affect the other components in the system.

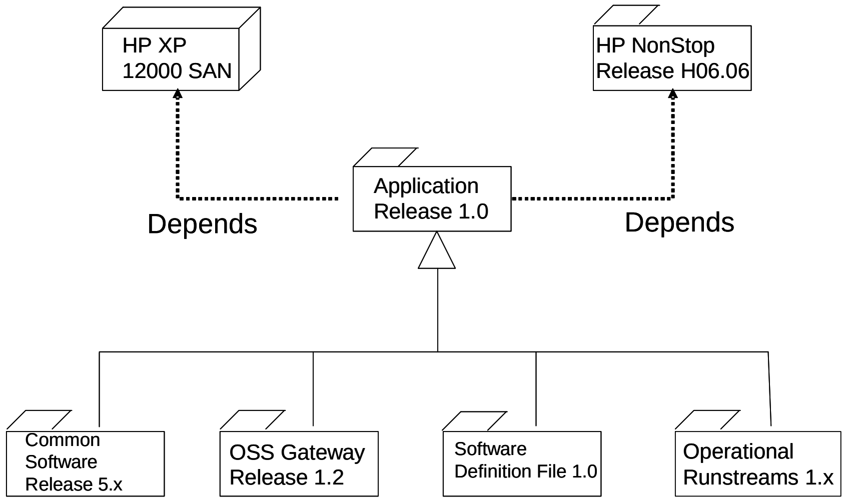


- *Use Case Diagrams* are used to show all operational and support procedures. They not only diagram standard procedures but also diagram the actions to be taken to recover from all exception conditions.

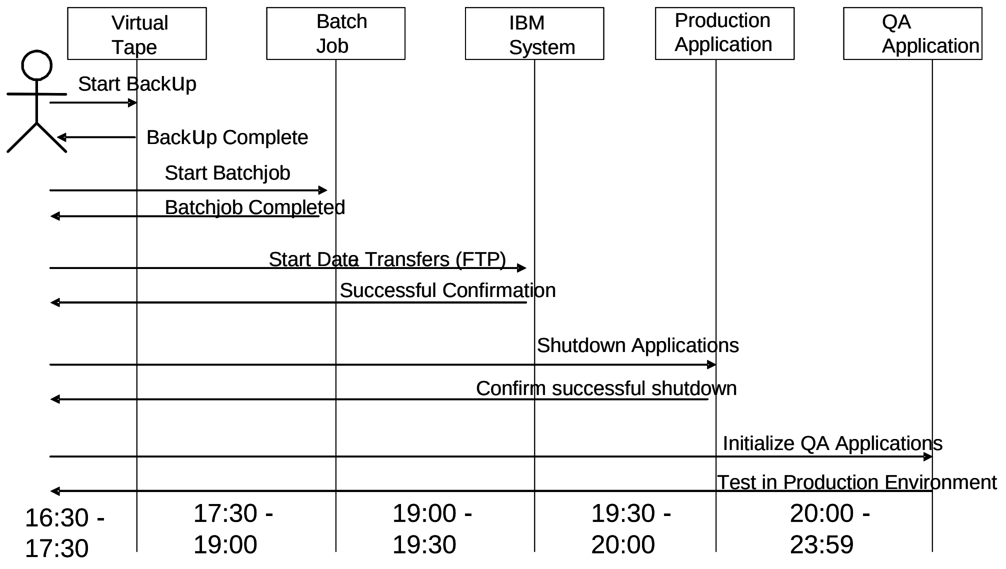


Installing a new software release in production for testing

- *Package Diagrams* are used to describe the contents of all releases.


HP XP 12000 SAN
HP NonStop Release H06.06
Application Release 1.0
Depends
Depends
Common Software Release 5.x
OSS Gateway Release 1.2
Software Definition File 1.0
Operational Runstreams 1.x

- *Sequence Diagrams* are used to diagram all operational procedures and to give an estimate of the time required.



Documenting Complex Algorithms

UML is intended to document processes (software, business, or otherwise) that can be represented by a set of objects. It specifically is not aimed at code documentation or algorithm diagramming (though activity diagrams can do this to some extent).

However, certain methods supported by an object may be quite complex and should be documented for future reference by developers and perhaps even by business staff. An excellent tool for this is the Nassi-Shneiderman chart.[2]

N-S charts are, in effect, an intuitive two-dimensional representation of modular psuedo-code. They are very easy to read. The two-dimensionality of the chart effectively reduces the effort to understand the algorithm by the square of its complexity.

N-S charts can be time-consuming to draw, but there is an excellent tool to generate them automatically from a psuedo-code representation of the algorithm. This tool is available from Robert Kast, a leading promoter of N-S charts, at bobk@legato.com.

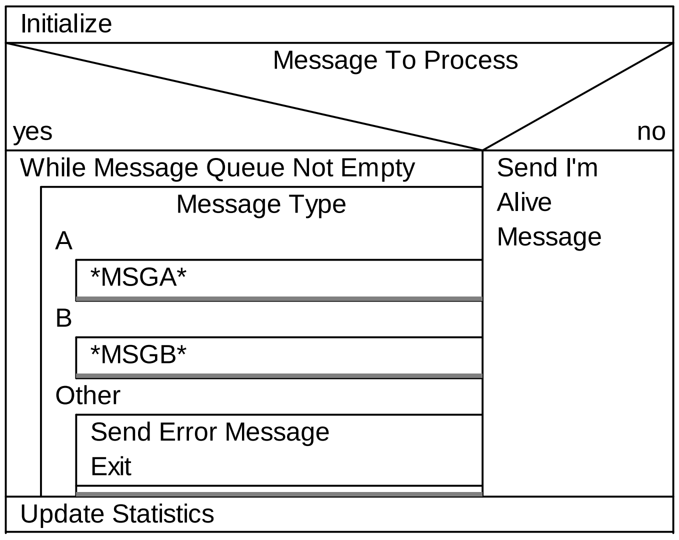
For instance, the following example of psuedo-code represents a somewhat simple algorithm:

```
Initialize
IF Message To Process THEN
        WHILE While Message Queue Not Empty DO
                SWITCH Message Type;
```

[2] See Chapter 15, Nassi-Shneiderman Charts, *Diagramming Techniques for Analysts and Programmers*, by James Martin and Carma McClure, Prentice-Hall; 1985.
See also the Sombers white paper, Structured Design with Nassi-Shneiderman Charts, at http://www.sombers.com/white_papers.htm).

```
                    CASE A;
                              *MSGA*
                    CASE B;
                              *MSGB*
                    CASE Other;
                              Send Error Message \l
                              Exit
                              EXIT
          END
     END
ELSE
     Send I'm Alive Message
END
Update Statistics
```

A click on a button yields the following N-S Chart:



In addition to documenting existing algorithms, N-S charts represent a valuable design tool to enforce structure and modularity and to ensure a consistent quality of design throughout a programming organization. In addition, these charts present an easily understandable depiction of data flow to the non-designers – the programmers and the users – and as such form an invaluable basis for maintenance documentation.

UML Products

We had said earlier that there were several good products to aid in the construction of UML diagrams. An updated list of these products is maintained by the Object Management Group on its web site, www.uml.org.

Summary

Documentation has always been a necessary evil, and too often the "evil" wins out over the "necessary." Documentation in many shops is often nonexistent or minimal.

The lack of good documentation, at least at a rudimentary level, is a must for proper operations and application maintenance.

Over the last decade, UML has become the accepted solution to solve this problem. It is an OMG standard recognized worldwide. As a result, many good off-the-shelf tools are now available to ease the creation and maintenance of UML documentation.

There should no longer be an excuse for not supporting the development, operations, and support staff with proper documentation.