

OBJECT CODE LOGIC ANALYSIS AND AUTOMATED MODIFICATION SYSTEM AND METHOD

This application claims benefit of Provisional application
Ser. No. 60/069,211 filed Dec. 11, 1997.

FIELD OF THE INVENTION

The present invention relates to the field of systems and
methods for the automated detection and remediation of
object code errors or anomalies, and in a preferred embodi-
ment to systems for addressing the so-called "millennium
bug" present in computer software systems.

BACKGROUND OF THE INVENTION

The software development process typically begins with
a statement of the intended functioning of a software
module, i.e., a statement of the problem. From this
statement, a high level analysis is performed to determine
the basic steps necessary to carry out the intended func-
tionality. These high level steps are then further analyzed and
translated into a computer language program. In many
instances, the computer program interacts with the computer
operating system, hardware, and/or other programs operat-
ing in the computer system.

Often, events occur after a final program is debugged,
compiled, and linked, which alter the operation of the
software, or make latent defects in the operational logic or
code apparent. One such instance is the so-called "Year 2000
Problem", or Y2K problem. This issue is insidious, because
existing and operational code with no present deficiency,
must be analyzed and appropriately replaced or remediated
before certain critical dates, or be at risk of failure. Since
many computer systems are interrelated and depend on each
other for operation, the failure of even small or remote
systems may lead to a "domino effect", resulting in failure
of related and more important systems. Some failure modes
are relatively minor, with inconvenient consequences, while
others are catastrophic, stopping operation of the computer
system, or worse, causing a failure of a real system con-
trolled by the computer system. Where the computer system
is an embedded controller or otherwise mission critical, the
software or firmware errors may lead to death or destruction.
For example, utility, elevator, flight control and even medi-
cal equipment and systems often have embedded controllers.
Even where the date information is ancillary to the main
function, a date reference which reveals a program logical
error may lead to failure of an entire system. Where this is
most apparent is where a system logs events or performs
trend analysis. If there is an inconsistency in dealing with
date data, the result could be a shutdown or erroneous
operation. In fact, legacy embedded systems may particu-
larly present this problem, because in the past, program
memory was at a premium and therefore conservation of
resources by compressing or truncating date information
was employed, even where this meant a critical limitation on
system design life.

As discussed in detail below, many stable computer
systems, particularly mainframe computer systems running
immense and complex code, will suffer from the Y2K
problem. This is a result of the use and reuse of legacy code
and the persistence of efficiency techniques and assumptions
which will no longer be valid early in the third millenium.
When confronted with this problem, two significant consid-
erations are the availability of accurate source code for the
software to be analyzed and corrected, and the testing and

debugging of replacement software to ensure that the func-
tionality is correct, the corrected software remains compat-
ible with systems to which it interfaces, and no new errors
are introduced. Another consideration is the time and
resources required to perform remediation, even where the
source code and testing environment are available.

There are a number of other, similar types of problems to
the Y2K problem. Essentially, the class of problems arise
because existing debugged compiled program code, for
which the source code may be unavailable or inconvenient,
or merely voluminous, becomes defective due to a relatively
rare occurrence of a definable event. Other examples include
program or operating system updates or partial updates,
desired substitution of elements which would not effect
fundamental program flow or logic, and translation of
parameter sets to ensure operation in a new environment.
Particularly, a problem often occurs in Microsoft Window-
style operating systems, e.g., Windows 3.X, Windows 95,
Windows 98, Windows NT, Windows CE, and other
variants, in which programs typically reference dynamic
link libraries or DLLs, visual basic objects or VBX, or other
code, which effectively becomes integrated with the oper-
ating system, and is often stored in a predetermined path
with such DLLs and VBXs from other programs. In this
case, where such common code is referenced by a single
name, it is possible, and even likely that an updated version
of the VBX or DLL by the same name will not operate with
older software, and newer software will not operate with an
older DLL or VBX. Similar problems occur in other cir-
cumstances and under other operating systems, mandating
synchronized updates of multiple system software compo-
nents.

Another instance of this problem is the potential rise of
the Dow Jones index above 10,000, which may lead to an
extra digit required for representing the value in existing
software which is otherwise fully functional. A further
instance is the change in currency units in Europe to the
ECU.

Precisely defined, the Year 2000 (Y2K) Problem is the
insufficient representation of the year datum as a 2 digit
field (or variable) in software applications and their associ-
ated data files and data bases, the incorrect calculation of
the leap year status, the inappropriate assumption of "1900"
or some other year as a base year or "1999" as the final
year, and the inaccurate programming of date calculations
with respect to these inaccuracies, including computations,
comparisons, assignments and other operations. The year
2000 is a leap year, unlike 1900. Normally century bound-
aries are not leap years; there are several exceptions to
this rule, one of them is if the year is divisible by 400
in which case it is a leap year. Identification of date
data and date calculations is complicated by the use of
pointers and arrays, obfuscated by the lack of standard
naming conventions and masked by embedding in other
data fields. As a result, the software in affected applica-
tions may incorrectly assume that the maximum legal
value of the year field is "99" or may incorrectly per-
form sorts and other operations that involve years des-
ignated by "00". Negative time duration could result
from subtractions from "00" (assumed to be year 2000).
Incorrect leap year calculations will incorrectly assume
that February 29th does not exist in the Year 2000. Thus,
in the year 2000, (or with some applications even earlier),
when the year field is incremented, many date dependent
computer algorithms that assume the year field will in-
crease monotonically, and will produce erroneous results
or cause abnormal program termination, with possible
disastrous consequences. Possible deleterious conse-
quences for affected applications range