Vornex Inc.
White Paper

5 Ways to Time Travel Test

A tester's guide to Temporal Testing

vornex

What is Time Travel Testing?

Time travel testing is the act of testing date and time sensitive functionality in order to validate business rules and logic that exists in a software stack. Other common names for this type of testing are

- Temporal Testing
- Time Shift Testing
- Forward Date Testing
- Date Simulation Testing

No matter the name, this type of testing is vital in discovering “date bugs” that exist throughout applications. These date bugs are platform and language independent and can exist in any application. Date bugs are most common in applications that rely heavily on date & time logic, such as financial, government, healthcare, energy, and insurance applications. Common test cases performed when time travel testing include:

- Enrollment
- Data Aging
- Insurance Rates
- Financial Validation
- Leap Year
- Daylight Savings Time
- End of Year, Month, Quarter
- Policy Forms & Wording
- Utility Rates
- Date & Time Fields
- Transaction Triggers
- Expiration Logic
- Tax Rates & Schemes
- Future Dates
- Billing Cycles
- Loan Repayment

Why Time Travel Test?

Time Travel testing falls in line with every other aspect of software quality assurance. Its purpose is to find defects, bugs, and errors with the goal of increasing software reliability, performance and functionality for the end user and company. This benefits your company as you reduce your exposure to the risk of production failures and down time which could result in potential loss of customers or revenue.

In addition, your organization may have critical governance, risk, and compliance initiatives to ensure optimal quality of customer facing software products. And depending on your industry, there may be government regulation requiring the testing of date sensitive functionality.

In the United States there exists a government insurance commissioner regulation that insurance companies must comply with in order to not be penalized. To meet this compliancy, companies must ensure their issued policies are completely accurate before issuing them. If date discrepancies occur and policies are out of compliance, penalties are issued against the insurer.

By incorporating temporal testing into your organizations software test cycle, you can fully ensure your critical date sensitive business rules are free of date bugs and your software is reliable.

Historical Bugs

When time travel is not incorporated into the testing cycle, bugs and software defects are introduced into the application and can cause varying amounts of

damage. When the damage is large enough, it can affect critical functionality and become news worthy. Below are historical examples of date bugs.

1) National Australia Bank's health payment system crashes by Leap-Year bug, *blocking 150,000 customers* from conducting medical transactions & processing payments. [1]

2) Microsoft's Cloud Platform Azure *offline for over 12 hours* due to Leap-Year bug. [2]

3) VMware ESX update introduces date-bug which blocks all virtual servers from starting up, *disabling customer's infrastructure.* [3]

4) Thousands of Bank of Queensland Point of Sale Terminals crash from rollover date-bug, *rejecting customer debit cards.* [4]

Moving forward, more critical dates will occur that require compliancy and testing. The most famous being the "Year 2038". This problem (similar to Y2K) is where UNIX's 32bit representation of time maxes out at the year 2038 and anything beyond will cause an overflow and roll back to 1901. This affects file systems, databases, languages, and other technology using 32-bit time representations. Thus, companies will need to time travel test in order to ensure 2038 compliancy.

Time travel is an important part of the quality assurance test cycle to ensure software reliability. In order to perform time travel testing an organization must pursue one of five options available today.

Option 1: Manually Shifting Server Time

This method involves manipulating the operating system clock via the system settings in order to time travel the server. Although straight forward on your personal laptop, in a large enterprise & corporate setting this is vastly time, cost, and resource intensive as no test automation is used. It typically requires

- Environment-Wide Coordination
- Management Approval
- *User downtime*
- System and Database Administer assistance
- Administrative permissions granted to local users, going against basic security practices

Once the manual time shift test is complete and the server time is set back to present time, a series of challenges and additional work exist. Specifically, your operating system files & *timestamps will be corrupted* with invalid dates, thus destroying backups, batch jobs, tasks, etc. This damage and the overall system confusion ultimately lead to the need to *restore your entire OS and application* back to their original state.

Each of these *can take hours to days*, all while users are offline and other testing is halted. Plus, you are limited to only testing one date-window at a time, slowing down your test cycle.

In addition, if the environment is contained within a domain authentication protocol, like Active Directory or Kerberos, then you cannot change the system clocks as these protocols rely on server times being in sync. If servers are not aligned within 5 minutes or less, servers get kicked off of the domain, thus blocking you from shifting time.

[1] http://www.smh.com.au/business/leap-year-blamed-for-hicaps-stumble-20120229-1u1z7.html
[2] http://www.wired.com/2012/03/azure-leap-year-bug/
[3] http://www.infoworld.com/d/virtualization/vmware-licensing-bug-blacks-out-virtual-servers-982
[4] http://www.smartcompany.com.au/growth/economy/12681-20100104-businesses-hit-by-bank-of-queensland-eftpos-bug.html

Pros
- No 3rd party tools required - Intuitive
Cons
- System Files & Timestamps Corrupted - OS & Application restores - Takes a large amount of time - Active Directory & Kerberos Lockout - Higher costs & resources required

These limitations lead companies to explore option 2.

Option 2: Isolate Servers

When companies utilize domain authentication protocols like Active Directory or Kerberos, testers are unable to manually shift server times without being kicked off the domain, thus blocking their testing. This leads to the option of purchasing additional servers, isolating them from the domain, and then manually shifting the server time. However, this approach requires the *increase of capital expenditure and server maintenance* as you scale your hardware & software in order to create a secondary environment dedicated to date shifting. In addition, with this new environment under a different security context, it differs from production and thus reduces the validity of your test cycle.

Pros
- No 3rd party tools required - Intuitive - Dedicated time travel environment
Cons
- Increased hardware and software - Increased deployment, maintenance, & human costs and resources - Test environment differs from production - Same challenges as “Manual Time Shift”

Option 3: Hardcoding

Hardcoding means utilizing the organization’s software engineers to hand-code a time travel test harness into the application code (such as with COBOL intrinsic functions or custom-stored database procedures). However, this approach has inherent disadvantages:

- *Increased lines of code* which equals increased development & maintenance costs plus the risk of introducing software bugs.
- QA degradation as testers are dependent on developers to do work out of their scope
- Struggle to ensure temporal consistency for distributed applications spanning multiple departments, environments, & platforms.

The biggest roadblock for hardcoding is the developers having access and availability to the source code of the entire software stack of the organization. For a multi-tier, distributed application stack containing popular 3rd party enterprise software like Oracle, SAP, WebSphere, SQL Server, etc., *attaining the source code is impossible* as they are proprietary to each vendor, thus preventing a developer from hardcoding a complete test harness.

Pros
- No 3rd party tools required - You own the time travel test harness - Built-in automation
Cons
- Increased lines of code   ○ Development & maintenance costs + risk of bugs - Time & resource drain for developers - Unable to time travel 3rd party apps

Option 4: Not Testing

Due to the complexities of trying to time travel an enterprise stack, an option is to simply not time travel test and hope date bugs do not exist in the application once in general release. However, the risk is too high for damage, making this *not a viable option* for companies who have shareholders, customers, and regulatory standards which expect the highest quality of product & service.

Pros
- No 3rd party tools required
Cons
- Risk of live date bugs - Possible application downtime - PR nightmare - Potential loss of customers, revenue, & shareholder confidence

These expectations and standards lead many companies to pursue option 5, test automation tools.

Option 5: Test Automation Tools

The easiest and best way to perform temporal testing is through a 3rd party software tool. They exist for all platforms like Windows, Linux, Solaris, AIX, HP-UX, HP-Nonstop, AS/400, z/OS. These tools provide efficient & instant time travel.

Pros
- Instant Time travel - Find and fix all date bugs - Expedite test cycle - Save time, money, & resources - Reduce hardware & software costs - Free up organizational resources. - Enable time travel in locked environments

Cons
- Cost

One of the leading tools available is TimeShiftX®.

Solution - TimeShiftX

TimeShiftX is a date and time simulation software that allows organizations to time travel enterprise applications and databases in the future or past to validate all date and time sensitive functionality. TimeShiftX provides “virtual times” to enable safe and instant time travel, thus expediting the validation of applications while reducing the resources and time needed to complete testing.

TimeShiftX enables time travel (even inside Active Directory & Kerberos) without changing system clocks, editing code, or isolating servers with features including…

- Instant Time Travel
  - No code changes, no altering server clocks, no isolating servers, no system impact. Just turn on TimeShiftX & begin temporal testing.
- Active Directory Compatibility
  - Safely time travel inside Active Directory, Kerberos, LDAP, and other domain authentication protocols.
- Secure time travel
  - No outside network calls performed
- Total App & DB Compatibility
  - TimeShiftX enables time travel for all applications & databases (SAP, SQL, Oracle, WebSphere, DB2, .NET, etc.)
- Cloud, Container, & Cross Platform Time Travel

    - TimeShiftX is compatible on all operating systems and can be run in containers like Dockers or in clouds such as Azure and AWS.
- Distributed Environment Time Shifting
    - TimeShiftX allows you to easily temporal test large, distributed software stacks from a single click.

TimeShiftX eliminates all the challenges with time travel testing and empowers companies to validate all date & time sensitive business rules instantly and seamlessly, thus increasing software reliability through automation while providing time, resource, and cost savings.

The TimeShiftX solution is deployed worldwide across clients who extend many industries including Banking, Finance, Healthcare, State & Local Government, Energy, and Insurance. Companies leveraging TimeShiftX include: Liberty Mutual, Lloyds Banking Group, Deutsche Bank, State of California, Nationwide Insurance, BNP Paribas, UnitedHealthcare, Mastercard, Gazprom and many others.

Companies choose TimeShiftX to meet deadlines, eliminate date bugs, save money, and reduce their exposure to the cost overruns and risk of unfound date bugs.

For more information

To learn more about TimeShiftX or to arrange a demo please visit: www.vornexinc.com or email info@vornexinc.com

vornex