



ERUDINE[®]
replaces legacy with flexibility



ERUDINE CANNOT BECOME LEGACY

WHITE PAPER BY DR TOBY SUCHAROV

Contents

EXECUTIVE SUMMARY	3
LEGACY ISSUES	4
LEGACY AND BEHAVIOUR	5
ERUDINE BEHAVIOUR ENGINE	6
AUTOMATIC CODE MODELLING	6
AUTOMATIC REQUIREMENTS MANAGEMENT	6
AUTOMATIC WORKFLOW AUDITING	6
AUTOMATIC DOCUMENTATION	7
LEGACY MANAGEMENT AND MAINTENANCE	7
TRADITIONAL CODE MAINTENANCE	7
Investigate areas of code impacted by change	7
Write report on suggested code changes needed	7
Write test scenarios for change	7
Apply changes to the code	8
Fix unintended changes in code and dependencies	8
Submit changes to code repository	8
ERUDINE CODE MAINTENANCE	8
Find test cases	8
Erudine executes all current rules	8
Edit the results to fulfil requirement change	9
Erudine detect exceptions in all test cases	9
Correct exceptions	9
Submit test case	9
Update rules, test cases and documentation	9
ERUDINE AND LEGACY	10
CONCLUSION	10
BIBLIOGRAPHY	11

© Remote Operations Limited (trading as Erudine) 2010

No part of this publication can be reproduced, stored in a retrieval system, transmitted or made available to the public in electronic form or by any other means (electronic, mechanical, photocopying, recording or otherwise) without the written permission of the publisher. Whilst every care has been taken to ensure the accuracy of the editorial content the publisher makes no representation and gives no warranty as to its accuracy and cannot accept any liability for any direct, indirect or consequential damage or loss howsoever caused arising out of or in connection with the content of this publication.

Erudine, Erudine Behaviour Engine, the Erudine logo and erudine.com are trademarks or registered trademarks of Remote Operations Limited (trading as Erudine) in the United Kingdom, other countries, or both. These and other Erudine trademarked terms are marked on their first occurrence in the information with the appropriate symbol (® or ™), indicating UK registered or common law trademarks owned by Erudine at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries.

The trademarks of other companies are marked on their first occurrence with the appropriate symbol (® or ™). Other company, product or service names may be trademarks or service marks of others.

In line with Erudine's environmental policy, this document has been optimised for double-sided printing.

EXECUTIVE SUMMARY

This is the second part of a series of white papers by Erudine that examines the issues and problems of Legacy Systems. First, this paper takes a look how legacy problems are usually confined to the components of a project that handle behaviour. Second, this paper introduces the Erudine Behaviour Engine which automates the key areas of the development process that contribute to legacy.

This paper builds on the ideas outlined in our first Legacy white paper *The Burden of Legacy* on how and why Legacy Systems are formed. These reasons are used to explain why projects built using the Erudine behaviour Engine can not turn into Legacy Systems

LEGACY ISSUES

The whitepaper *The Burden of Legacy* discussed how Legacy Systems are created and concluded that the legacy occurs when the developers or maintainers of a system lacked a sufficient mental model of the code to alter the code base. Such situations arise when:

- The developers who wrote the system leave the company.
- The system becomes too complex or large to understand.
- The system is inherited or commissioned by an external service provider.

The bleak view is that due to the size and complexity of modern IT projects all such systems will become Legacy Systems. The Bowman Project, the British Army's new geographic information system to manage communications, testifies to this statement. It has been described as a future Legacy System in recognition of the fact that it is expected to become legacy shortly after its launch.

LEGACY AND BEHAVIOUR

There are numerous packages that aid developers in building IT projects. Off the shelf graphical and database packages can be used for the user interface and persistence layers.

The business layer is the area of the project which is most susceptible to legacy due to the fact that it is the least supported by commodity software and, by definition, requires the most customization.

The business layer can be divided into its separate areas of work:

- **Transactions:** This is the safe modification of information from one state to another. Bank and invoice systems are particularly interested in transactions.
- **Scalability:** This is the ability to increase performance under an increased load when more resources are added.
- **Security:** This is the infrastructure of the project that ensures privacy and protection of information.
- **Behaviour:** This is the code that processes the information relevant to the project.



The first three areas are well supported by off the shelf solutions, well documented protocols and international standards. As such these areas of the projects should be much easier to produce and maintain.

However, the behaviour module of the project is usually a bespoke solution, with little support from packages and protocols. For this reason the behaviour module of the project is the weakest link in a standard IT project and is nearly always the first area where legacy issues become apparent.

ERUDINE BEHAVIOUR ENGINE

Erudine is a high level application that allows developers to build complex IT projects using a large number of automated features aimed at solving the critical issues that create legacy systems.

Erudine is a behaviour engine that automatically generates a suite of test cases as requirements are added to a project. The power of this technique is further explained in the whitepaper Unlocking Tacit Knowledge. The behaviour engine with its automated test cases strongly supports developers building the behaviour module of a modern IT application providing:

- Automatic Code Modelling
- Automatic Requirements Management
- Automatic Workflow Auditing
- Automatic Documentation

AUTOMATIC CODE MODELLING

Using the Erudine Behaviour Engine the developer will never have to worry about how his current changes will affect the overall structure of the code. The Erudine Behaviour Engine maintains a model of the code making sure new requirements will not unintentionally alter old functionality.

Using Case Based reasoning Erudine automatically generates a suite of tests as new behaviour is added to the system. These tests are used to automatically validate any future requirements that are added to the system. Any contradiction between new and old are automatically resolved by Erudine or, when the answer is non-obvious, Erudine will present the developer with the appropriate test cases and ask for a justification for the change in behaviour for these cases.

New developers will be able to understand how an Erudine project works much more rapidly, as the test cases will support any future changes he makes to the system.

An Erudine project also means that a developer can confidently alter more complex projects than was previously possible, as the developer is not able to break old functionality when making changes. Erudine will detect and force resolution of any behaviour clashes.

AUTOMATIC REQUIREMENTS MANAGEMENT

Addition of requirements towards the end of a project is inevitable. In fact the ability to quickly add new requirements into an existing project is essential to allow companies to exploit changing market situations. Erudine's requirements management is driven by its automatic code modelling. Code modelling allows new requirements to be added without unintentionally altering previous requirements.

The Erudine Behaviour Engine automates all stages of requirements from entry, to their removal or change. As requirements management is one area that is very likely to turn modern well managed and staffed projects into legacy, automation in this area is critical to solving legacy problems for large projects.

AUTOMATIC WORKFLOW AUDITING

In complex workflow projects, it is very important that the flow of data through the system is understood and can be reliably tested. Any changes that alter the flow of data through the system can be detected.

Erudine projects can track data as it moves through project modules and create an audit trail. When future changes are made to the project the full set of audit data can be run through the system which will check, at several different levels, whether data passes through the system correctly.

Using Erudine workflow auditing, code maintenance and adding new requirements can be done with confidence knowing that any unintended changes in functionality will be detected by validating the set of previous audits, which describes the workflow of data through the system.

AUTOMATIC DOCUMENTATION

Functionality covered within the Erudine Behaviour Engine is automatically documented as a by-product of building the suite of test cases. Each test case represents part of the behaviour of the system. A developer can use the Erudine Behaviour Engine's test suite as documentation to view the requirements of the behaviour engine. The suite's test cases are also the cases on which the developer originally captured this requirement. This allows the developer to understand the context in which the test was captured.

Test suites provide the developer with complete documentation as well as contextual information about why the requirements were added. This is an immensely powerful tool for new developers who seek to understand an Erudine project or an old developer to simply refresh their memory on the working of the system.

LEGACY MANAGEMENT AND MAINTENANCE

TRADITIONAL CODE MAINTENANCE

The pain caused by Legacy Systems is very apparent when considered in terms of the code maintenance process.

The process starts when the developer is given a specification of requirements that needs changing in their Legacy System.

Investigate areas of code impacted by change

The developer undertakes an analysis of the areas of code and code dependencies that they think will be affected by the requirements change.

Write report on suggested code changes needed

The developer writes a brief report on his suggested solution to the requirements change and submits this to a committee of programmers (which could simply be the lead code maintenance programmer) for approval.

Write test scenarios for change

The developer writes a series of tests which could be in the form of actual code or a written work document that accurately captures the area of code the developer is changing to allow the developer to verify the correct working of the system after he has made his changes.

Apply changes to the code

The developer implements the changes as outlined in his report.

Run tests and find areas of code and dependencies impacted by change: The developer runs his tests and then runs any more general tests that make up part of the system quality assurance process.

Fix unintended changes in code and dependencies

The developer reviews the unintended changes that his new code has caused and either fixes the new areas of code, fully integrating his changes or returns to the start of the process and rethinks his initial code fix.

Write documentation detailing changes: The developer writes documentation detailing the changes he has made to the code, noting any dependencies and new functionality added to the system.

Submit changes to code repository

The developer submits his code changes to the main code repository and the changes are integrated into the system.

This process could take anywhere from a day to several weeks to implement depending on a large number of factors including the scale of any changes made, concurrent projects and other developers having current ownership of code blocks.

After changes have been made to the system the developer will often not be confident that his modifications have not caused unintended bugs elsewhere in the code which were not envisaged as part of the code maintenance process.

It should also be noted that the process described here is idealised, it is not uncommon for the documentation stage to be poor due to the developers' poor knowledge of the legacy application.

ERUDINE CODE MAINTENANCE

Similarly, the Erudine process starts when the requirements that need changing are delivered to the developer.

Find test cases

A changing requirement will have cases or data onto which this change will impact. The developer finds this set of cases for entry into the system as part of Erudine's Case Based Reasoning with Conclusion and Justification process.

Erudine executes all current rules

Erudine executes its current rules on the case, at this stage the conclusion obtained from the system will be incorrect as no changes have been made to the system.

Edit the results to fulfil requirement change

For each new test case the developer needs to decide what the new conclusion will be. The developer need only submit the new outcome or conclusion to the Erudine Behaviour Engine at this stage.

Erudine detect exceptions in all test cases

The Erudine Behaviour Engine finds the set of previous cases which are impacted by the changes made by the developer, these cases could be in contradiction or similar enough to the new requirement to require clarification as to why the requirements are different. If possible these found cases are resolved automatically by Erudine, otherwise these cases are presented to the user for manual resolution through the graphical interface.

Correct exceptions

The developer resolves the cases which have been sent by the Erudine Behaviour Engine for manual clarification.

This process typically involves the developer justifying the difference between the current case and previous test cases to help it differentiate between similar requirements

Submit test case

Once everything is correct the developer submits the test case to the Erudine Behaviour Engine.

Update rules, test cases and documentation

The Erudine Behaviour Engine updates its rules, test cases and documentation automatically.

This entire process will take about 10 minutes to half a day to undertake, the most time consuming step may well be finding or generating the set of test cases that satisfy the requirement being added.

Due to Erudine's automation of the key processes of maintaining code the developer can be confident that the changes made are accurate and the functionality of the code has not been unintentionally changed. The developer also has the benefit of strong documentation and a full test suite describing all the system behaviour.

ERUDINE AND LEGACY

Projects built using the Erudine Behaviour Engine are not prone to the legacy issues of traditional projects, as the code maintenance process is automated within the Erudine Behaviour Engine. This automation allows Erudine Behaviour Engine built projects to be altered much faster and more reliably than traditional projects.

However, the benefits of Erudine are not just applicable to new projects. Current Legacy Systems can be rebuilt with the aid of Erudine Legacy Mode, a feature specifically designed for the conversion of Legacy Systems into Erudine projects.

CONCLUSION

As the complexity of IT projects increases the problems associated with Legacy Systems are becoming much more prevalent. We have seen that it is the bespoke behaviour of the business layer that is hardest to maintain. The stringent code maintenance process required for Legacy Systems clearly reflects this.

Erudine is a new technology for handling large projects, and represents a discontinuous advance in the way that IT projects are built and maintained. By automating the construction of the behaviour module within the project, Erudine ensures that a project cannot become a Legacy System.

BIBLIOGRAPHY

Crossing the Chasm, Geoffrey A. Moore, Capstone Publishing Ltd, ISBN: 1841120634, 1999.

WEB DOCUMENTS

1. DARWIN: On Incremental Migration of Legacy Systems. Michael Brodie, Michael Stonebraker.
<http://db.cs.berkeley.edu/papers/S2K-93-25.pdf>
2. CIO Update: AD Survey of Enterprise IT Managers Reveals Top Concerns. Joseph Feiman. Gartner Research: <http://www3.gartner.com>
3. CIO Update: Legacy Modernization Magic Quadrant Helps in Providing Applications for Tomorrow. Dale Vecchio Gartner Research: <http://www3.gartner.com>
4. Chickens and Turkeys Migrate, but not Necessarily in IT. Ben Wilson
<http://www.anubex.com/migrationscentre!chickensandturkeys.asp>
5. Issues and Challenges Facing Legacy Systems. Federico Zoufal.
http://www.developer.com/mgmt/article.php/11085_1492531_2
6. Applications Intelligence Survey. Hal Knowledge Solutions Hal Knowledge Solutions White Paper.
<http://www.halks.com>
7. Legacy Integration. Something old, something new: integrating Legacy Systems. Bob Sutor.
http://www.ebizq.net/topics/legacy_integration/features/5229.html

info@erudine.com
+44 (0)8456 123 862
www.erudine.com

Central House
Beckwith Knowle
Otley Road
Harrogate
North Yorkshire
HG3 1UF

