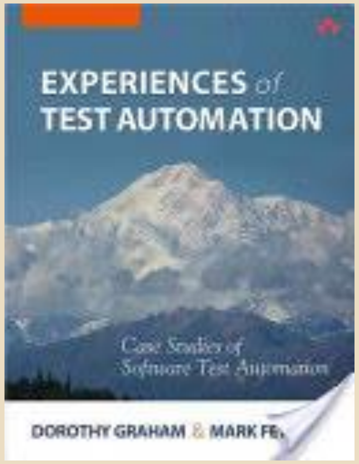
SOFTWARE TEST AUTOMATION SUCCESS
A CASE STUDY



Mike Snyman

CONTENTS

Introduction

- Define test automation
  - “ Automation is any use of tools to aid or enable testing”
- Test automation is not new
  - What is new is the concept of a “non-programming dedicated tester”
- Test automation is the key
  - Incremental innovation

Introduction

- *“ The first rule of any technology used in a business is that automation applied to any efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency.”*

Bill Gates

“ A fool with a tool is still a fool.”

Grady Booch

Challenges

1. Lack of an end-to-end defined automation process.
2. Insufficient re-use of tools, scenarios and data.
3. Automation involved late in the process.
4. No living documentation.
5. Slow uptake of test automation and shelfware.
6. No means of proving automation benefits and cost justification.
7. Dependency on key resources.

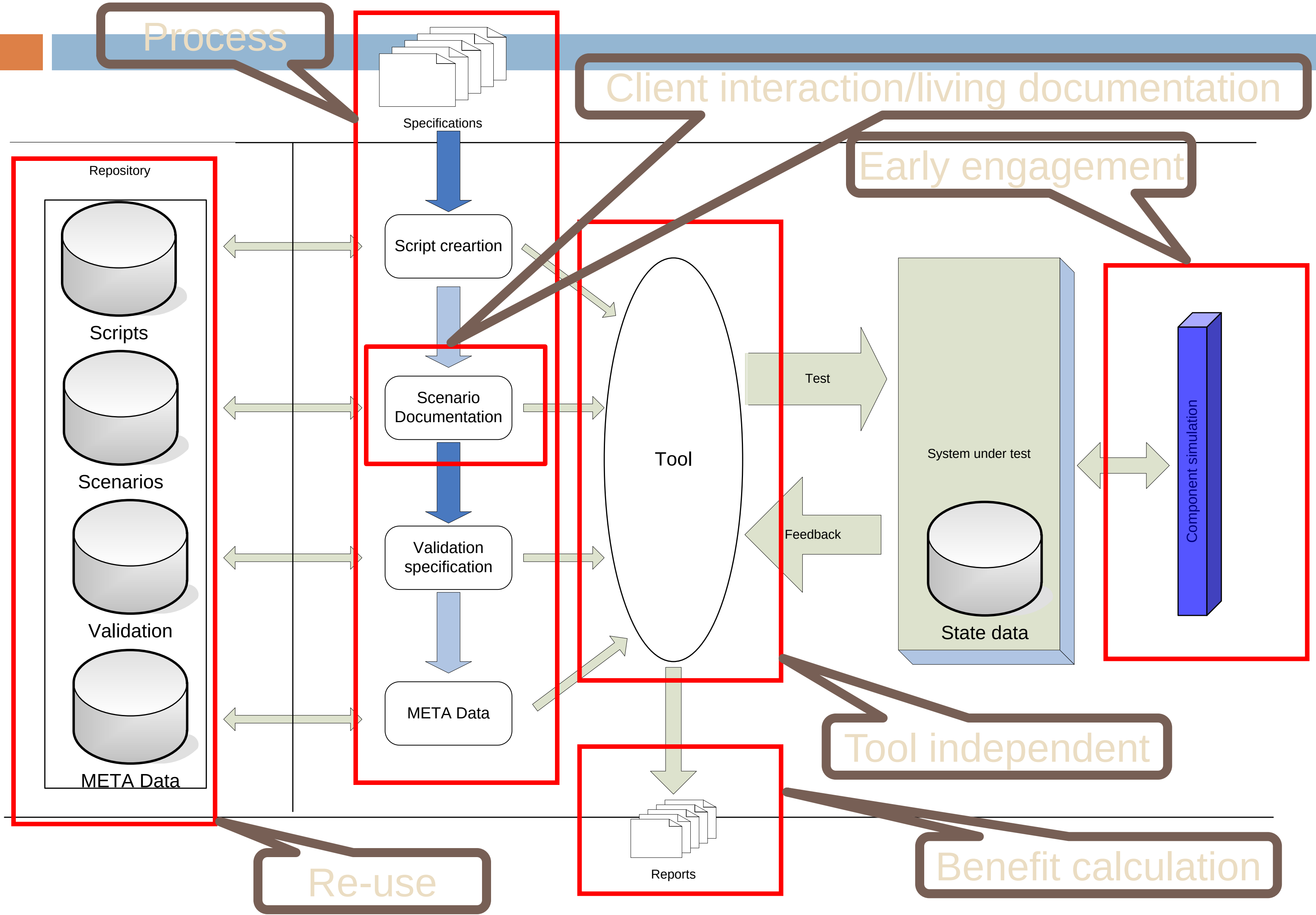
Solving the problem

1. Formal organisational project launched.
2. Project had to be based on a sound financial base.
3. Funding for the project approved at the highest level.
4. Failure not an option.
5. Resourced with capable and skilled resources.
6. Driven by a passionate person.

Automation framework

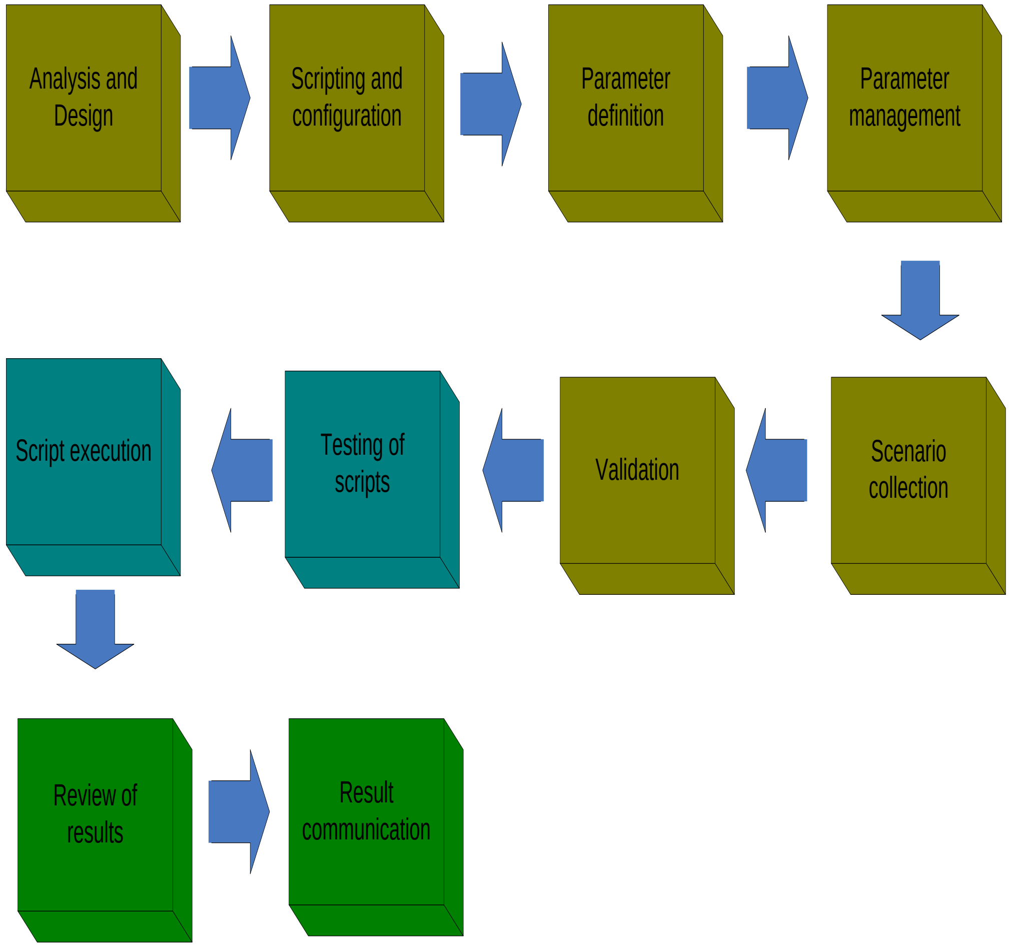
- Is a set of assumptions, concepts and principles
- Graphical depiction easily translated into solutions
- High-level frameworks
- Low-level frameworks

High-level Automation framework



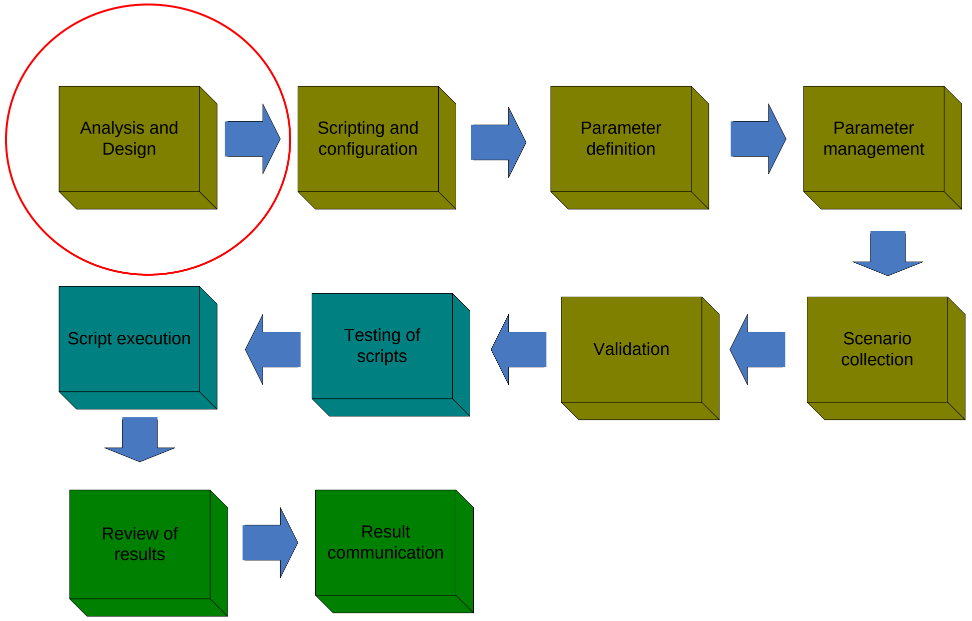
Automation process

- Simple
- Flexible
- Detailed enough for estimation
- Integrated
- Real



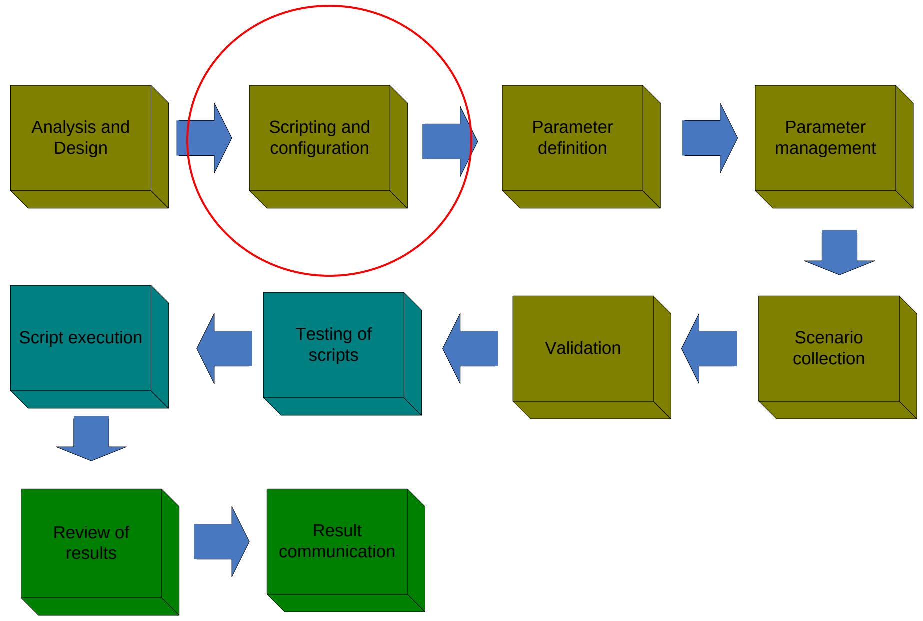
Analysis and design

- Understand the client's test requirements.
- Automated solution is proposed.
- Prototyping.



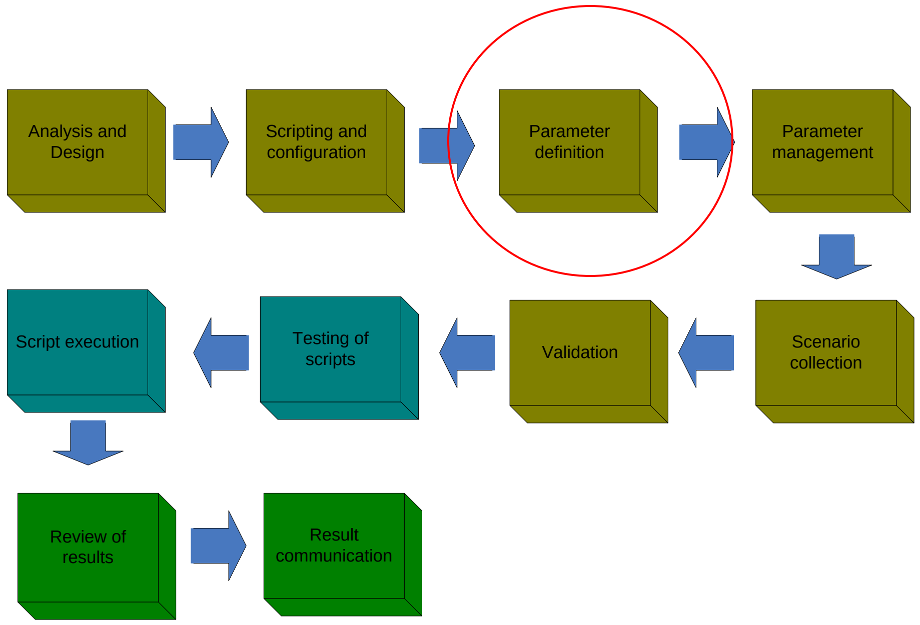
Scripting/Configuration

- User requirements implemented.
- Activities in this phase could include recording, scripting and building of special utilities.
- Internal testing of automated test solution.



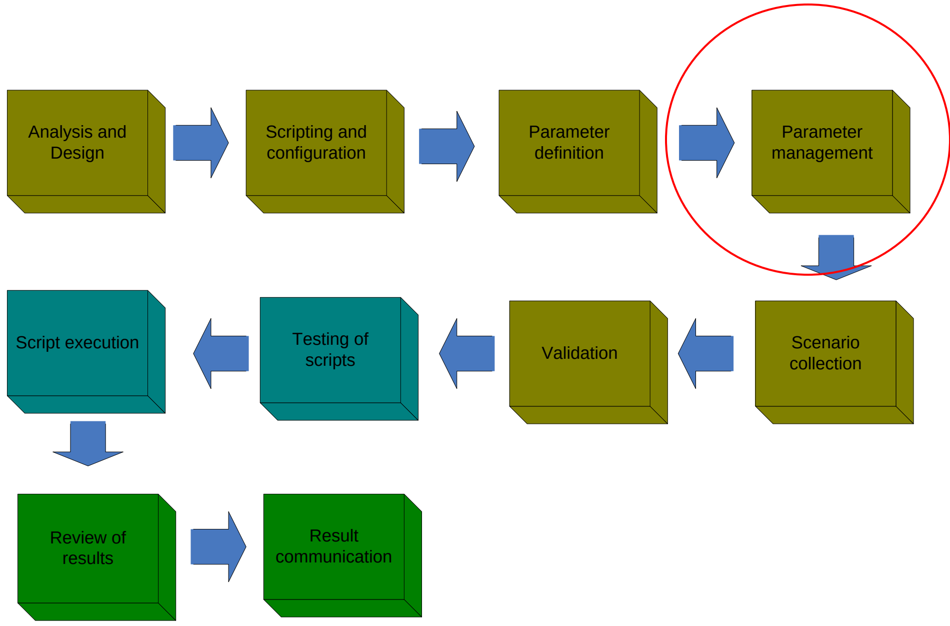
Parameter identification

- Scripts assessed against user-defined scenarios.
- Components identified for parameterisation.
- Components categorised based on nature and function.



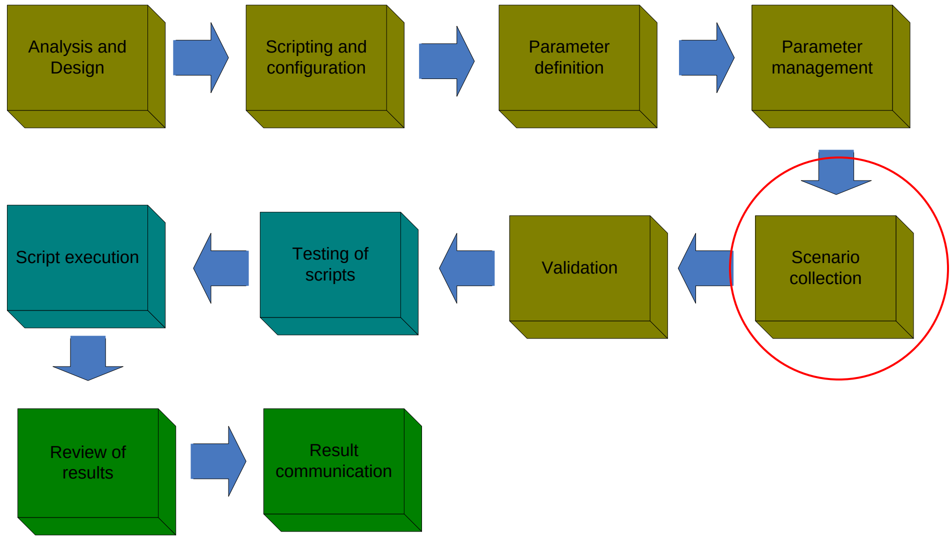
Parameter management

- Large amounts of data created in previous step are managed.
- Customised spreadsheets are created.
- Requirement exists that data in these sheets can be maintained by non-technical personnel.
- All scenarios described in both technical and non-technical terms.



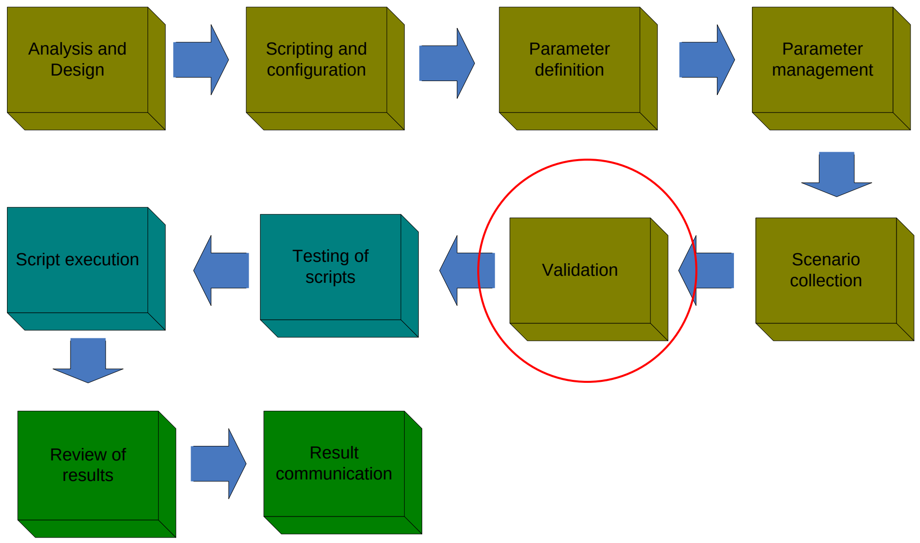
Scenario collection

- Spreadsheet populated with scenarios provided by stakeholders of the system.
- Manual test cases could provide input.
- Actual production outages used as input.



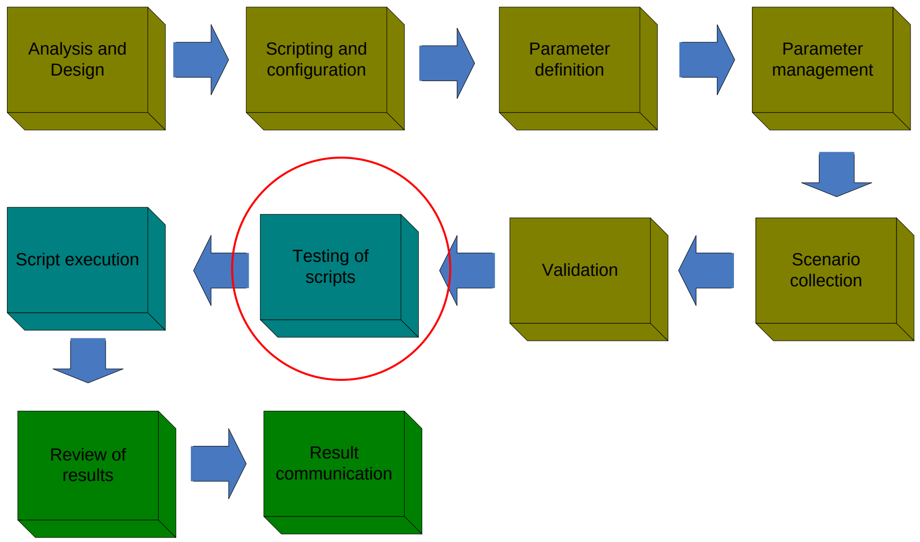
Validation

- Automated validation of results important due to volume.
- Spreadsheet updated with expected results.
- Validation done by script using information from spreadsheet.
- Clear “Pass” or “Fail” indicators provided.
- Summary of results provided.



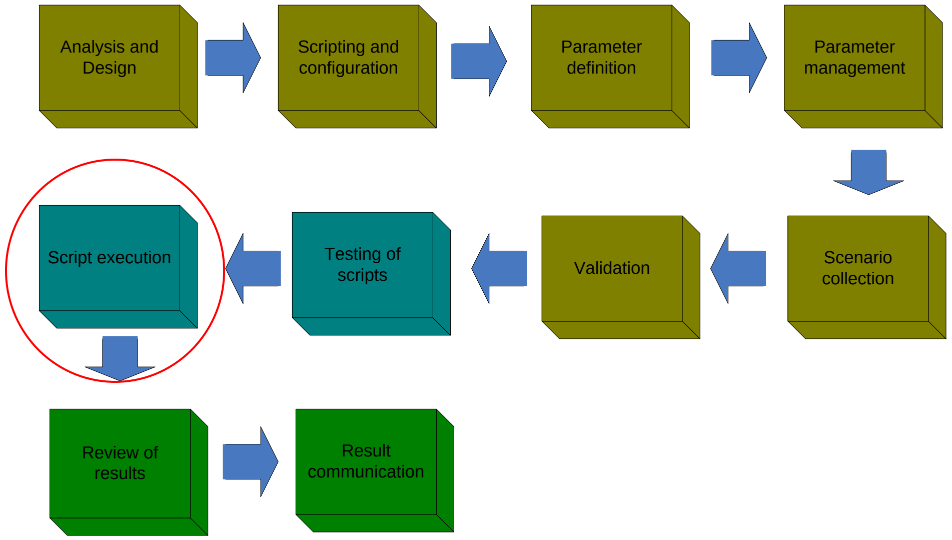
Testing of scripts

- Like all new software, the scripts must be tested.
- Important that when defects are reported during the actual test cycle that the tool is above reproach.
- Scripts must be able to react to anomalies in a predicted fashion.
- Must still be able to report effectively on operational scenarios.



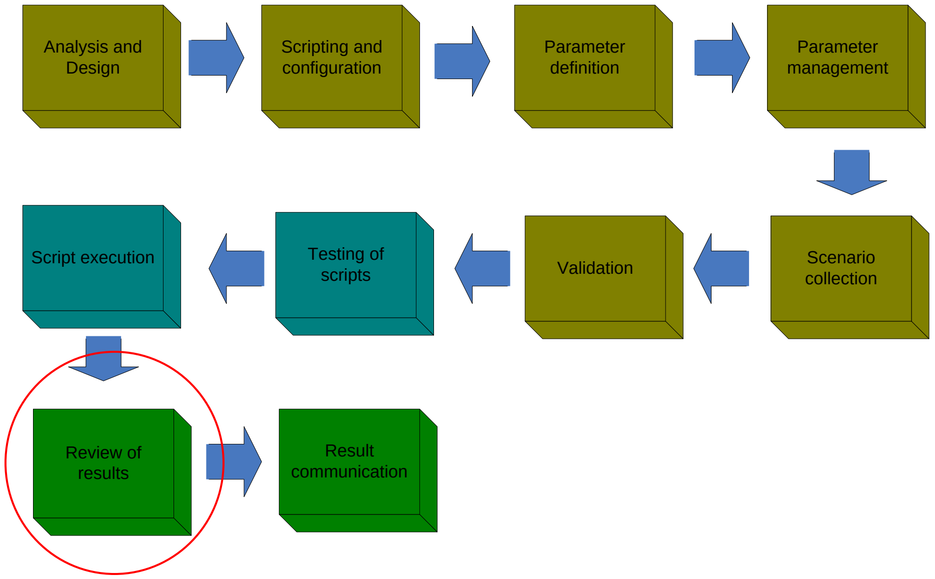
Script execution

- Scripts are run against target application.
- Stability of system determines support required.
- Strive to provide as much meaningful data in the shortest possible time.



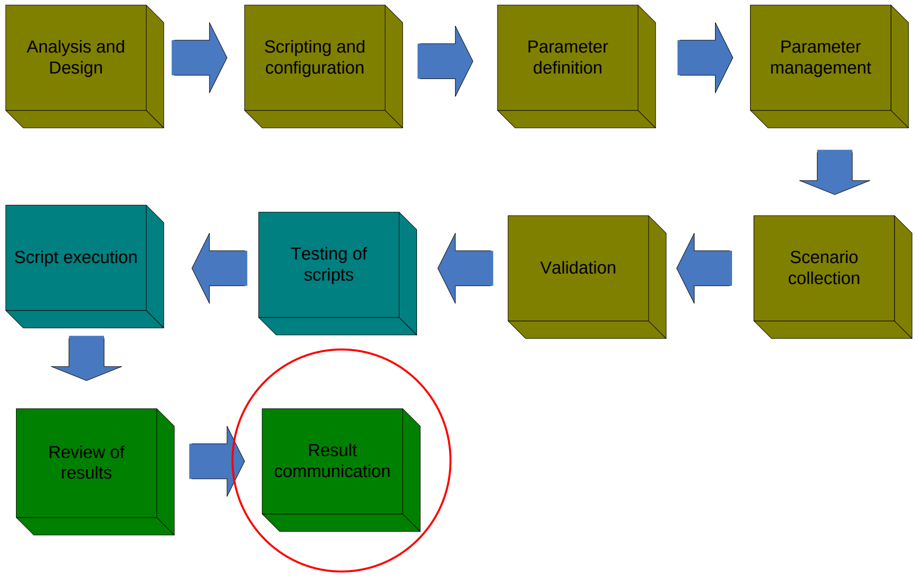
Review of results

- Results are reviewed internally.
- Focus on abnormal failures due to environmental or other conditions.
- Checks are done to ensure data integrity and scope coverage.

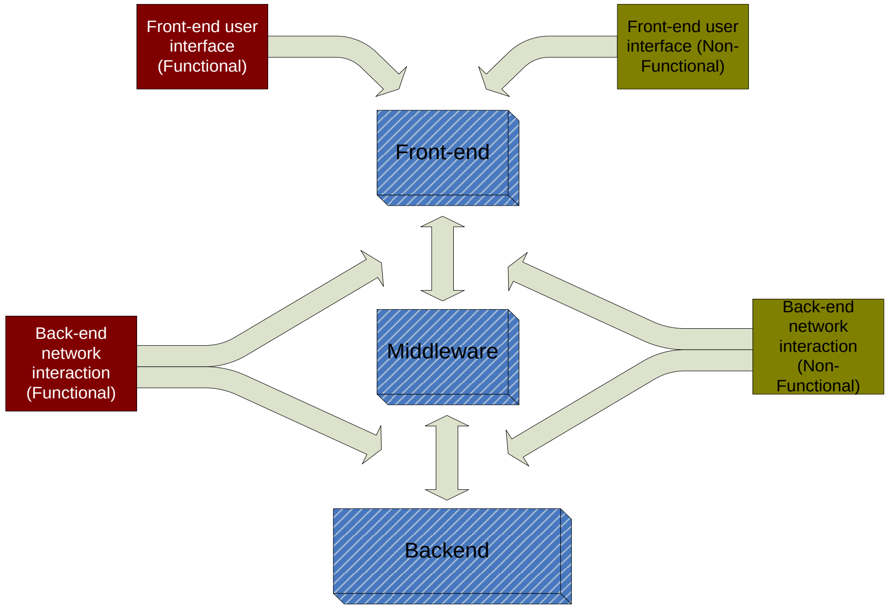


Result and benefit communication

- Because of size, results are saved in repositories.
- Screen dumps of all defects are kept.
- Communicate benefit in terms of automation lifecycle stage.



Categorising the toolset

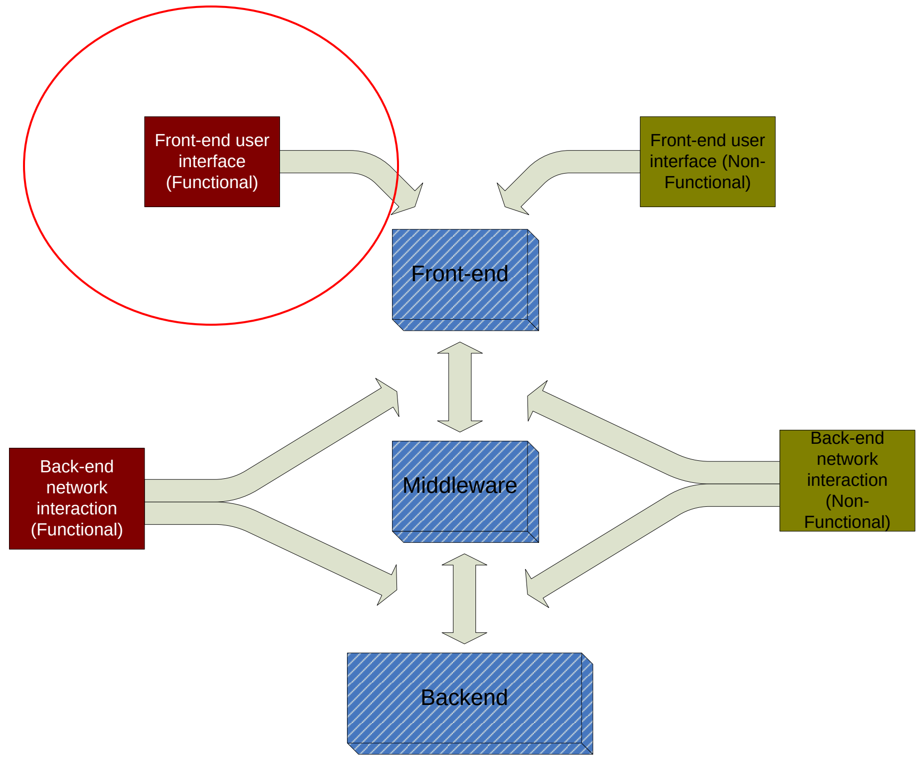

Front-end user interface (Functional)
Front-end user interface (Non-Functional)
Front-end
Back-end network interaction (Functional)
Middleware
Back-end network interaction (Non-Functional)
Backend

Why?

The strength in categorising tools is the ability to provide uniquely customised, automated testing solutions which, in conjunction with manual testing, aim to mitigate both the product and project risks associated with solution deployment.

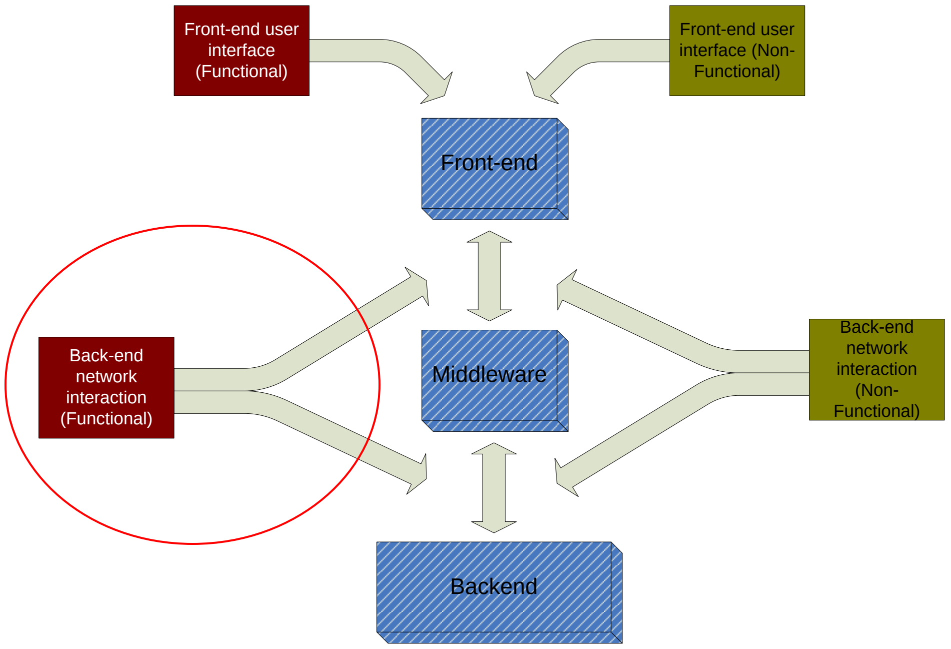
Front-end user interface (Functional)

- Record and playback.
- Simulate user interaction with applications.
- Provide support for unit, integration, system and acceptance testing.
- User regression testing particularly suits the nature of these tools.


Front-end user interface (Functional)
Front-end user interface (Non-Functional)
Front-end
Back-end network interaction (Functional)
Middleware
Back-end network interaction (Non-Functional)
Backend

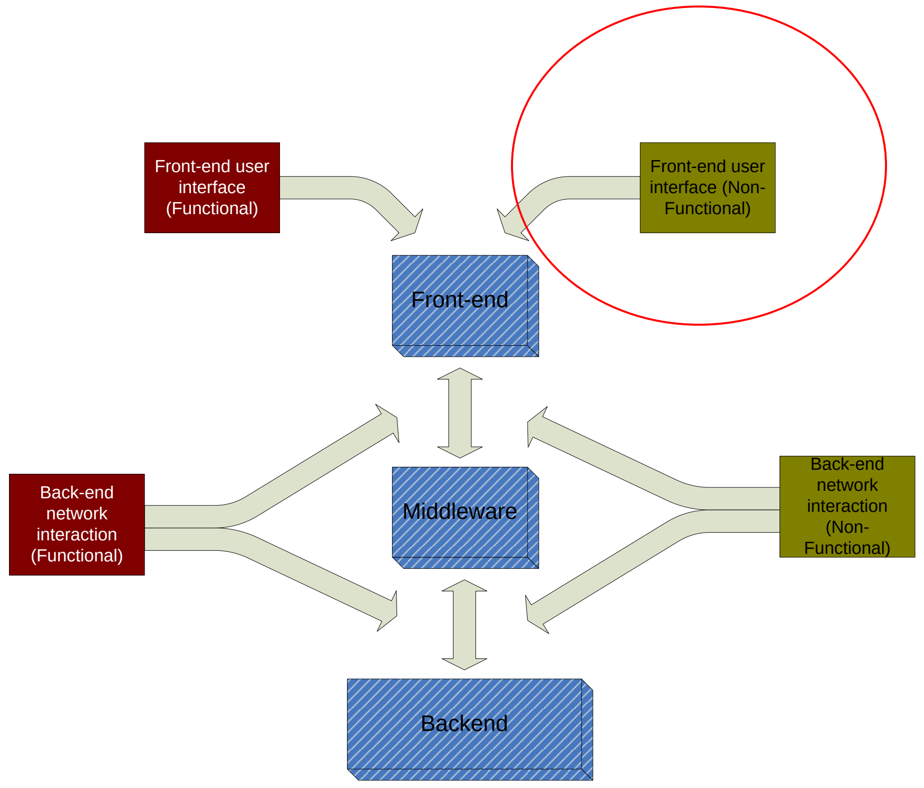
Back-end or network interaction (Functional)

- Ability to simulate user interaction in the absence of a front-end.
- Supports bottom-up integration.
- Provides the ability to find defects much sooner in the SDLC.
- We found it valuable in testing SOA implementations.


Front-end user interface (Functional)
Front-end user interface (Non-Functional)
Front-end
Middleware
Back-end network interaction (Functional)
Back-end network interaction (Non-Functional)
Backend

Front-end user interface (Non-functional)

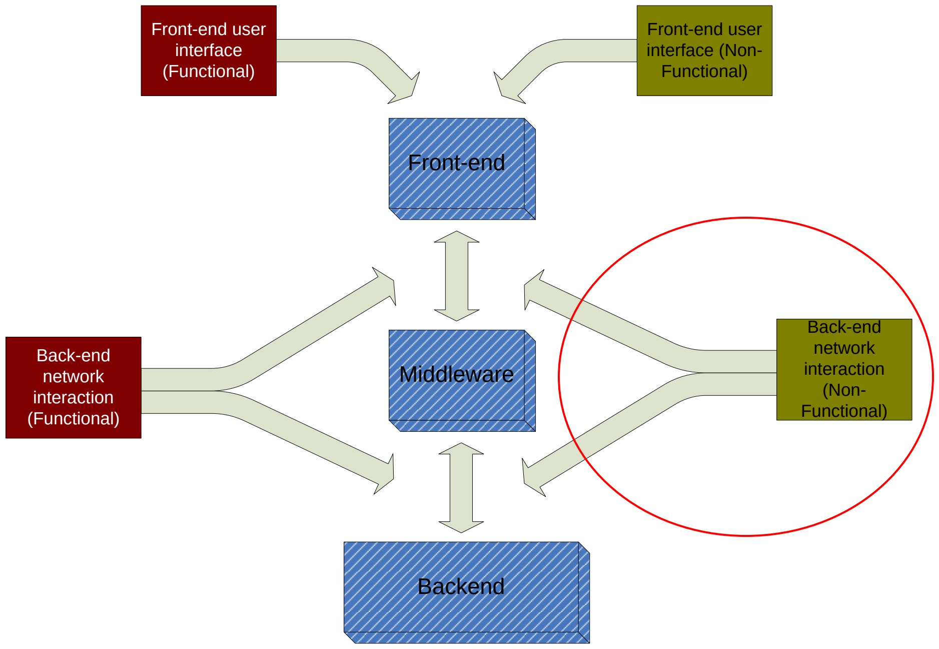
- Ability to generate the required load for performance, load and stress testing.
- These types of tests can typically not be performed manually.
- Tool is dependent on the functional quality of the application under test.
- Test environment comparison with production.



Front-end user interface (Functional)
Front-end user interface (Non-Functional)
Front-end
Back-end network interaction (Functional)
Middleware
Back-end network interaction (Non-Functional)
Backend

Back-end or network interaction (Non-functional)

- Due to the nature of implementation, it is not practical to use the front-end to generate the required load (ATM, point-of-sale devices).
- Typically these kinds of tests cannot be performed manually.
- Test environment comparison with production.
- We found it valuable in testing the performance of services in SOA implementations.


Front-end user interface (Functional)
Front-end user interface (Non-Functional)
Front-end
Middleware
Back-end network interaction (Functional)
Back-end network interaction (Non-Functional)
Backend

General automated testing utilities

- Provides general support for manual and automated testing activities.
- Compares large amounts of data.
- Generates unique data sets from production data.
- Generates vast amount of test data.
- Provides support to all previously-mentioned tool categories.

Automation benefit calculations

1. Historically problematic to get accurate measures.
2. Focus initially on quality.
3. Difficult to quantify quality in monetary terms.
4. Decision taken to focus on productivity improvement and associated cost reduction.

Automation benefit calculations process

1. Identify the smallest common element.
2. In most cases it would be the parameters identified in the scripting process.
3. Compare effort (duration) associated with manual testing with that of automated testing per parameter.
4. Describe the improvement in monetary terms by using an average resource cost.

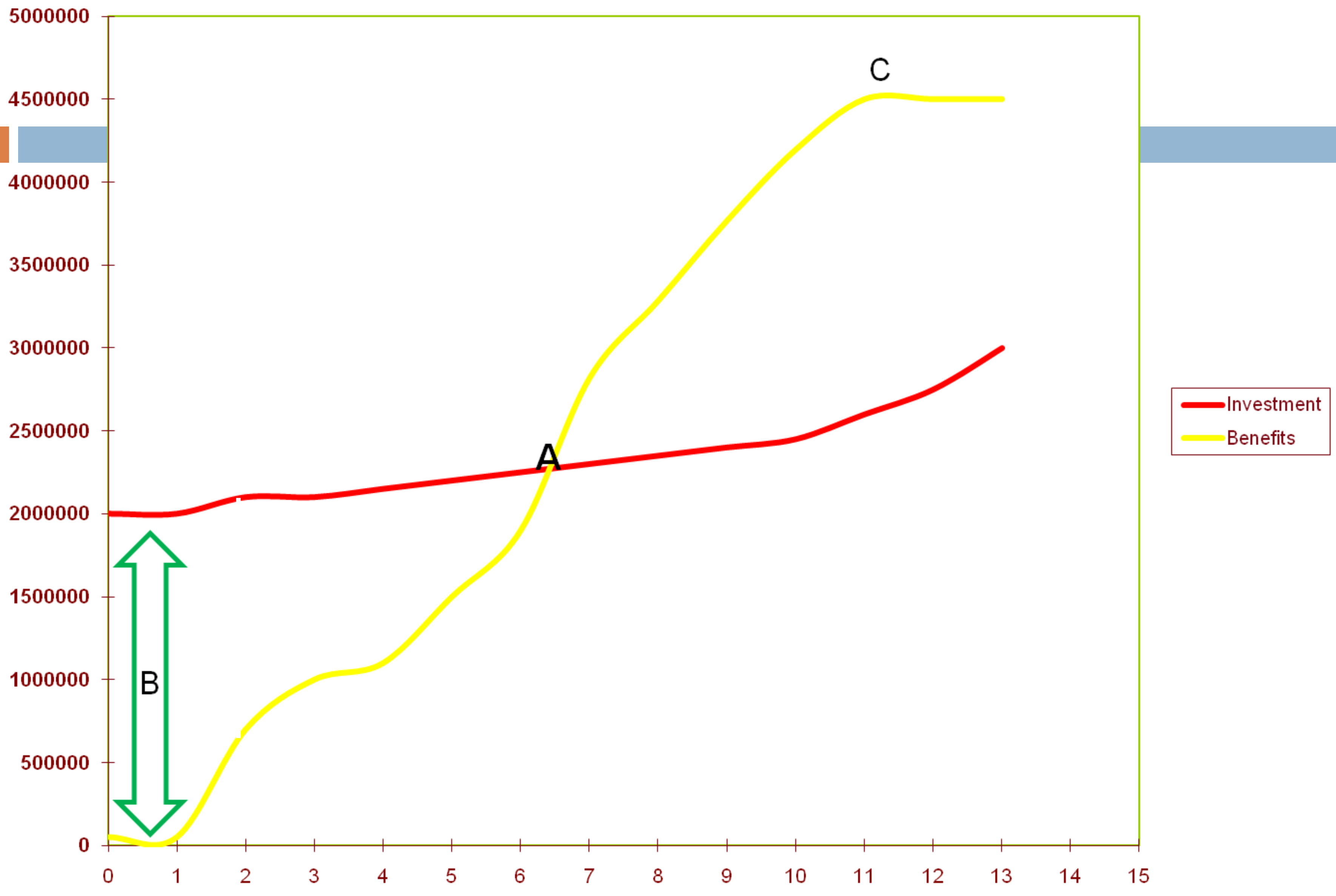
Automation benefit

1. Total cost saving for the period of 3 years (5 Systems):
   R86 183 171.00 ($ 8600 000 US)

2. Benefit confirmed by organisational stakeholders, system owners and users.

3. Calculation method reviewed by Finance.

4. Return of this nature justified the significant investment made in both project testing and automation tools.



Regression cycles

What have we learned?

1. Having a test tool is not a strategy.
2. Automation does not test in the same way that a manual tester does.
3. Record and playback is only the start.
4. Automated test scripts are software programs and must be treated as such.
5. The value lies in maintenance.
6. Categorise your toolset, and design integrated solutions.

.

Thank you

Mikes@Nedbank.co.za