A Case Study in UI Test Scalability

or

Leveraging Test Case Architectural
Design to Promote Reuse and Scalability

Agenda

- Background of Vistaprint
- Goal Statement
- Discussion of Principles:
  - Test Object model
  - Reusable Test Artifacts
  - Test Parameters
  - A "Test to Software" Taxonomy
- Benefits of adoption

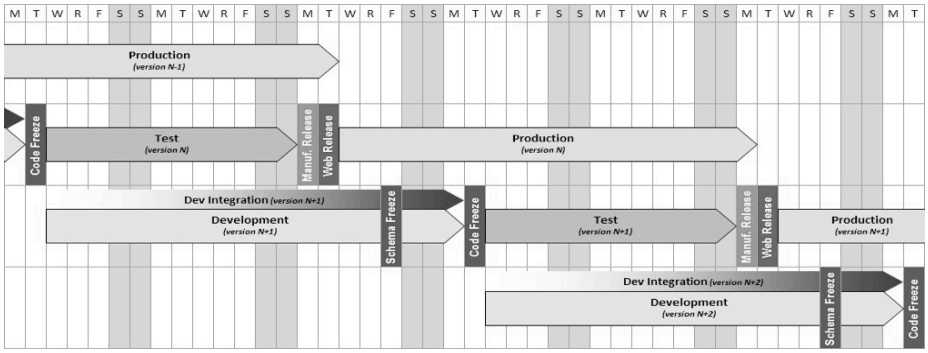
BACKGROUND OF VISTAPRINT

Background of VP Business

Vistaprint is an e-commerce marketing company with over 200 products in 25 localized websites.

Background of VP Lifecycle

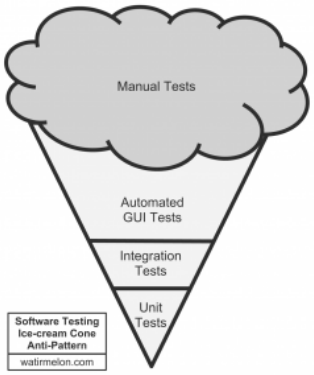
Vistaprint releases its entire codebase into production every 3 weeks.
Our legacy codebase lacks many testability principles, so most testing is done through a full environment UI interface.



Background of VP Quality

Initially our test documentation process was disjointed and flaky. It was ineffective at handling our organic growth, and maintenance costs increased constantly.

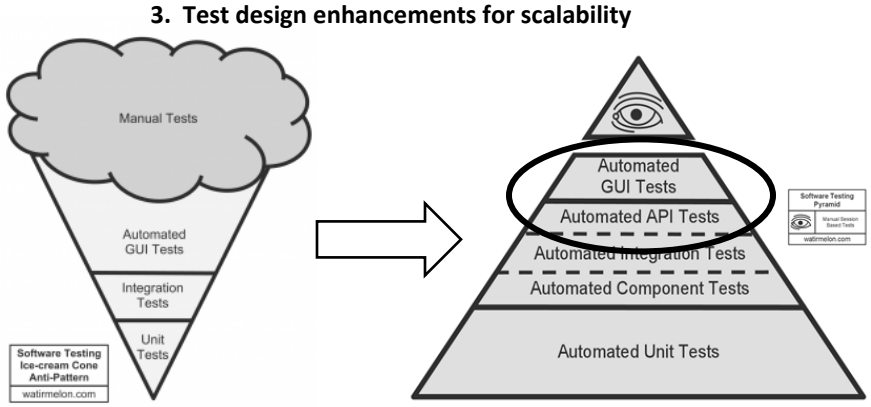
1. Test tools and documentation were duplicated and fragmented
2. Vast range of skillsets and processes across QE teams
3. Organic growth highlighted ineffectiveness in our test design



VP Quality Improvement Model

We have embarked on a three-part initiative to increase the quality and scalability of the quality organization's test asset suite. This presentation focuses on the third:

1. Quality platform and tooling
2. Quality engineering training and skills improvement
3. **Test design enhancements for scalability**



Discussion of Principles

GOAL STATEMENT

What was the goal?

The goal of this effort was to develop a test design structure to increase optimization of our test documentation and scripting, as well as reduce long term maintenance cost.

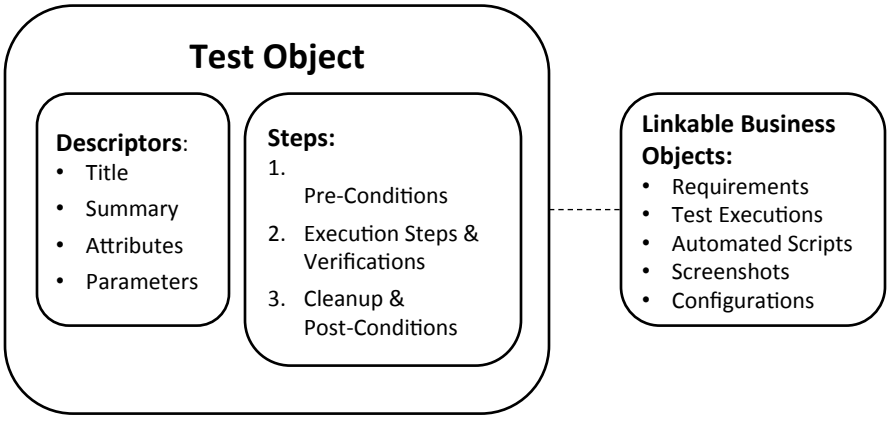
Defining the Principles

- We first identified certain infrastructural principles that were important for large-scale UI test management:
  - A clear structure
  - Isolation
  - Reusability
  - Flexibility
  - Some level of consistency across teams

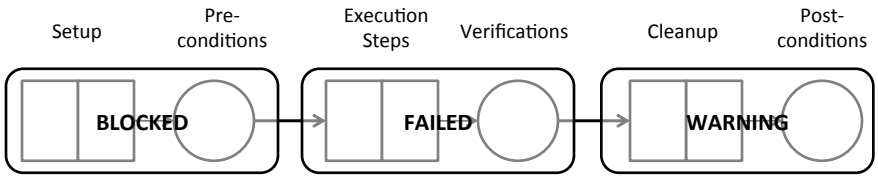
Discussion of Principles

THE TEST OBJECT MODEL

Defining the Model



Test Step Labels



Categorizing test steps in such a manner gave us several benefits:

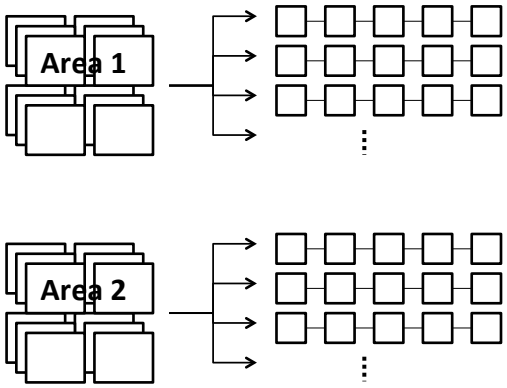
1. Deeper visibility into how the environment is acting as a whole
2. Visibility into dependencies and relationships across areas
3. Control over automation failures for different step types
4. Setup templates can be used to help streamline documentation
5. Errors can be contained, allowing for better error tracking

Discussion of Principles

REUSABLE TEST ARTIFACTS

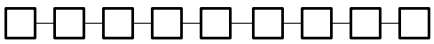
Reusable Test Artifacts

Building Blocks are individual, parameter driven feature actions.

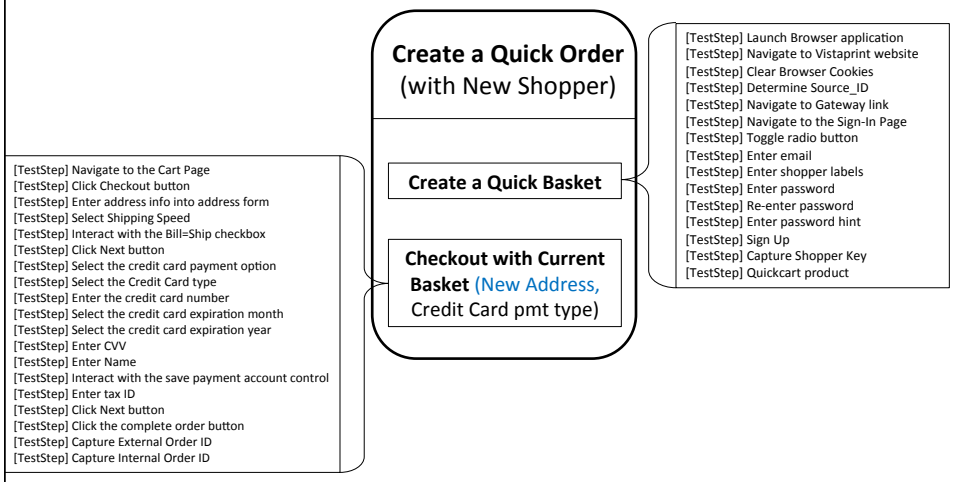


Templates are ordered sets of building blocks linked together as system paths.

Templates can be structured in many ways



Template Example



Benefits of Test Artifacts

The adoption of these artifacts has:

1. Introduced isolation into our test suite
2. Minimized test creation time
3. Optimized scripting and documentation maintenance
4. Allowed domain knowledge to be contained to domain experts

Test Repository metrics:

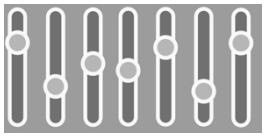
36k Total steps
20k steps call artifacts
2.5k Distinct artifacts

- ~55% of all steps are canned objects
- ~87% of all canned steps are repeats
- ~50% of steps need no documentation

Discussion of Principles

TEST CONFIGURATION

Parameters and Configurations



Parameters allow for tests to be configured in different ways using the same general steps.

It was important that parameters in building blocks and templates automatically pass to any calling object.

Defining open-ended variables to have values configured at a later stage allowed many similar scenarios to be achieved in a single test outline.

This has shown benefits in test management, as well as increasing coverage efficiently.

Parameter Example

Create a Quick Order
(with New Shopper)

This template will provide 22 different parameters to the calling test for free!

Call <**Create a Quick Basket**> with the following parameters:

Server = ?,
Locale = ?,
Environment = ?,
Browser = ?,
PPP-SourceID = ?,
FeatureValues = ?,
ShopperLabels = ?,
email = ?,
CookieType = ?,
passwordhint = ?,
retypepassword = ?,
password = ?,
combo_id = ?,
quantity = ?,
pf_id = ?

Call <**Checkout with Current Basket, Capture Order IDs** (New Address, Credit Card pmt type)> with the following parameters:

OTO_button = ?,
Checkout button = ?,
Carrier = ?,
Speed = ?,
Address = ?,
CC_type = ?,
CC_number = ?

Configuration Example

Create a Quick Order
(with New Shopper)

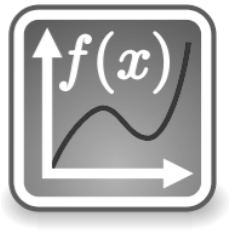
- Browser testing
  - All Defaults
  - Set 'Browser' to:
    1. Internet Explorer 7
    2. Internet Explorer 8
    3. Internet Explorer 9
    4. Internet Explorer 10
    5. Chrome 35
    6. etc
- Payments testing
  - All Defaults
  - Set 'CC_type', 'CC_number' to:
    1. VISA, 4111
    2. VISA, 4321
    3. Mastercard, 5555
    4. Mastercard, 5431
    5. etc

Parameters and Configs - Learnings

- While parameters can exist around almost anything, knowing when to use the highest value parameters is key.
- Parameters allow us to configure data inputs to tests without needing to change the steps or script. This empowered many more testers to be able to easily manage test suites.
- The need for overriding parameter values for whole test sets forced us to create a tool to allow for this specific requirement.
- Data Driving was not a feature that was given enough thought. This should definitely be considered in a final product.

Runtime Test Variables

- Not all useful data is known prior to runtime. We also needed the ability to capture information during runtime, store it, and refer to it later or output it to a log.

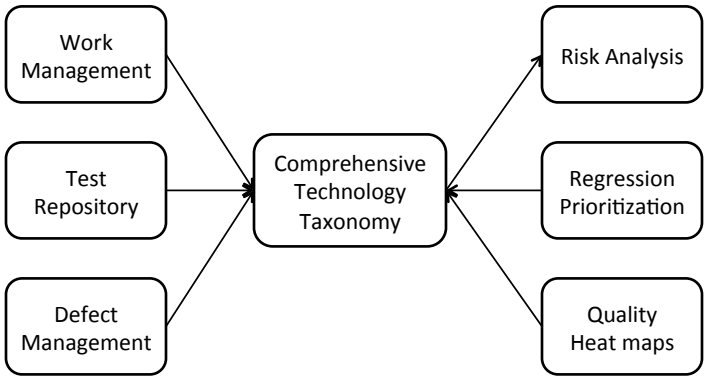


- Test variables allowed this to be possible, and like parameters, if these values are created in a template, they are inherited by the calling test.

Discussion of Principles

A "TEST TO TECHNOLOGY" TAXONOMY

A Technology Taxonomy



Our technology organization has developed a component taxonomy skeleton of the whole codebase. This takes over 17 million lines of code and compartmentalizes them into roughly 500 components.

Leveraging our Taxonomy

This is currently a 4 layer taxonomy, identifying:

business programs → sub-programs → technology groups → components

Leveraging this taxonomy across company tools opened the door to several new quality enhancements:

- A mapping between release updates and regression tests
- Ownership for all levels of the regression test hierarchy
- Increased metrics for Release Risk analysis
- Better context for Defect Management metrics

SUMMARY OF BENEFITS

Summary of Benefits

- Isolation and Reusability
- Documentation Consistency
- Test Suite Flexibility
- Optimized triage of test results

Questions?