

Exploratory Test Automation: Investment Modeling as an Example

Cem Kaner, J.D., Ph.D.

Executive Vice-President, Association for Software Testing

Professor of Software Engineering, Florida Institute of Technology

Immune IT

October, 2009

Copyright (c) Cem Kaner 2009

These notes are partially based on research that was supported by NSF Grant CCLI-0717613 "Adaptation & Implementation of an Activity-Based Online or Hybrid Course in Software Testing." Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Overview

1. Testers provide empirical research services, exposing quality-related information to our clients.
2. We play a major role in determining:
 - How valuable our research is for our clients
 - How much our clients will value our work
3. We can increase our value, if we can:
 - Find problems that have greater impact on the business, or
 - Use technology to find problems that are hard to find by hand
4. It's hard to discuss testing-value in depth in software testing courses or books:
 - Deeper testing requires product knowledge. It can take a long teaching time to build enough product insight for a student tester to understand tests at that level
 - Exploratory test automation architectures call for deeper levels of technical sophistication than we can reach in most testing courses.

Software testing

- is an empirical
- technical
- investigation
- conducted to provide stakeholders
- with information
- about the quality
- of the product or service under test

We design and run tests in order to gain useful information about the product's quality.

Empirical? -- All tests are experiments.

Information? -- Reduction of uncertainty. Read Karl Popper (Conjectures & Refutations) on the goals of experimentation

Testing is always a search for information

- Find important bugs, to get them fixed
- Assess the quality of the product
- Help managers make release decisions
- Block premature product releases
- Help predict and control product support costs
- Check interoperability with other products
- Find safe scenarios for use of the product
- Assess conformance to specifications
- Certify the product meets a particular standard
- Ensure the testing process meets accountability standards
- Minimize the risk of safety-related lawsuits
- Help clients improve product quality & testability
- Help clients improve their processes
- Evaluate the product for a third party

Different objectives require different testing tools and strategies and will yield different tests, different test documentation and different test results.

Test techniques

A test technique is a recipe, or a model, that guides us in creating specific tests. Examples of common test techniques:

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- User testing
- All-pairs combination testing
- Data flow testing
- Build-verification testing
- State-model based testing
- High volume automated testing
- Device compatibility testing
- Testing to maximize statement and branch coverage

We pick the technique that provides the best set of attributes, given our context and the objective of our search

Techniques differ in how to define a good test

Power. When a problem exists, the test will reveal it

Valid. When the test reveals a problem, it is a genuine problem

Value. Reveals things your clients want to know about the product or project

Credible. Client will believe that people will do the things done in this test

Representative of events most likely to be encountered by the user

Non-redundant. This test represents a larger group that address the same risk

Motivating. Your client will want to fix the problem exposed by this test

Maintainable. Easy to revise in the face of product changes

Repeatable. Easy and inexpensive to reuse the test.

Performable. Can do the test as designed

Refutability: Designed to challenge basic or critical assumptions (e.g. your theory of the user's goals is all wrong)

Coverage. Part of a collection of tests that together address a class of issues

Easy to evaluate.

Supports troubleshooting. Provides useful information for the debugging programmer

Appropriately complex. As a program gets more stable, use more complex tests

Accountable. You can explain, justify, and prove you ran it

Cost. Includes time and effort, as well as direct costs

Opportunity Cost. Developing and performing this test prevents you from doing other work

Exploratory software testing

- is a style of software testing
- that emphasizes the personal freedom and responsibility
- of the individual tester
- to continually optimize the value of her work
- by treating
 - test-related learning,
 - test design,
 - test execution, and
 - test result interpretation
- as mutually supportive activities
- that run in parallel throughout the project.

The essential goal of exploration is learning.

As with all strategies for learning, we can learn at a deeper level or a more superficial one; we can learn critically relevant information or trivia; we can learn broadly or narrowly.

QuickTests

A **quicktest** (or an attack) is a cheap test that has some value but requires little preparation, knowledge, or time to perform.

- A quicktest is a technique that starts from a theory of error (how the program could be broken) and generates tests optimized for errors of that type.
- Like **any** test technique, quicktesting may be more like scripted testing or more like ET
 - *depends on the mindset of the tester. (ET is a style of testing, not a technique)*
- This is a great tactic at the start of the project, but if it is your whole project, you miss the issues that are unique to the particular application you are testing.

Some history

- Participants at the 7th Los Altos Workshop on Software Testing (Exploratory Testing, 1999) pulled together a collection of these.
- Al Jorgensen & James Whittaker developed a series of attacks, published in Whittaker's *How to Break Software*.
- Elisabeth Hendrickson teaches courses on bug hunting techniques and tools, many of which are quicktests or tools that support them.

Classic Example of a QuickTest

Traditional boundary testing

- All you need is the variable, and its possible values.
- You need very little information about the meaning of the variable (why people assign values to it, what it interacts with).
- You test at boundaries because miscoding of boundaries is a common error.

Note the foundation of this test.

There is a programming error so common that it's worth building a test technique optimized to find errors of that type.

"Touring" as an exploratory learning activity

- The analogy of exploration to touring was described / taught beginning in the 1990s by Elisabeth Hendrickson, Mike Kelly, James Bach, Mike Bolton and me. Think of it as functionally similar to a structured brainstorming approach--excellent for surfacing a broad collection of ideas, that we can then explore in depth, one at a time.
- The "tour" is a themed, usually superficial, exploration of a product, a risk, or a context
- Example: in a Feature tour, you work through an application to discover all of its features and controls
- Tours can be done alone, or in a tour group, and they might benefit from a tour guide (think of training new testers in exploratory testing)
- For several links, see <http://www.developsense.com/2009/04/of-testing-tours-and-dashboards.html>
- List on next page, see <http://www.michaeldkelly.com/blog/archives/50>

"Touring" as a learning activity

Feature tour: Move through the application and get familiar with all the controls and features you come across.

Complexity tour: Find the five most complex things about the application.

Claims tour: Find all the information in the product that tells you what the product does.

Configuration tour: Attempt to find all the ways you can change settings in the product in a way that the application retains those settings.

User tour: Imagine five users for the product and the information they would want from the product or the major features they would be interested in.

Testability tour: Find all the features you can use as testability features and/or identify tools you have available that you can use to help in your testing.

Scenario tour: Imagine five realistic scenarios for how the users identified in the user tour would use this product.

Variability tour: Look for things you can change in the application - and then you try to change them.

Interoperability tour: What does this application interact with?

Data tour: Identify the major data elements of the application.

Structure tour: Find everything you can about what comprises the physical product (code, interfaces, hardware, files, etc...).

Commodity-Level Software Testing

You are a commodity if:

- your client perceives you as equivalent to the other members of your class

Commodity testers:

- have standardized skills / knowledge
- are easily replaced
- are cheaply outsourced
- add relatively little to the project

Commodities

There are green bananas
and ripe bananas
and rotten bananas
and big bananas
and little bananas.
But by and large,
a banana is a banana.

Commodity testers have little on-the-job control over their pay, status, job security, opportunity for professional growth or the focus of their work.

Commodities and Certification

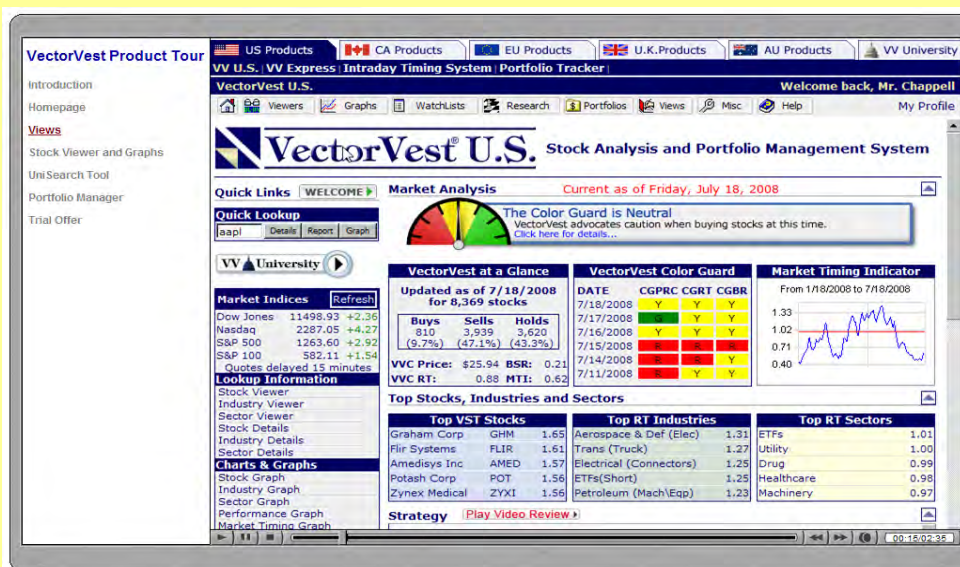
- A three-day course and an easy test won't distinguish professionally skilled testers from tester-commodities
- Simplified, prescriptive ("you should do it this way") training:
 - given to everyone
 - gets everyone thinking the same way
 - easily replaced commodities

What level are you working at? (Some Examples)

CHECKING	<ul style="list-style-type: none">• Testing for UI implementation weakness (e.g. boundary tests)• Straightforward nonconformance testing• Verification should be thought of as the handmaiden to validation
BASIC EXPLORATION	<ul style="list-style-type: none">• Quicktests• Straightforward tours to determine the basics of the product, the platform, the market, the risks, etc.• Here, we are on the road to validation (but might not be there yet)
SYSTEMATIC VARIATION	<ul style="list-style-type: none">• Conscious, efficiently-run sampling strategy for testing compatibility with big pool of devices / interoperable products / data-sharing partners, etc.• Conscious, efficiently-run strategy for assessing data quality, improving coverage (by intentionally-defined criteria)
BUSINESS VALUE	<ul style="list-style-type: none">• Assess the extent to which the product provides the value for which it was designed, e.g. via exploratory scenario testing
EXPERT INVESTIGATION	<ul style="list-style-type: none">• Expose root causes of hard to replicate problems• Model-building for challenging circumstances (e.g. skilled performance testing)• Vulnerabilities that require deep technical knowledge (some security testing)• Extent to which the product solves vital but hard-to-solve business problems

Testing focused on Value to the Business: An Example

Non-professional investors often rely on advisors, and tools, hoping to beat the market by following the advice



(NOTE: I use VectorVest in several examples because I liked it enough to research it more carefully than its competitors.

Despite my critical comments, you should understand that this product offers significant benefits, especially in the accessibility of its highly detailed historical fundamentals data.)

http://www.youtube.com/watch?v=lb_h_mwKk-o

<http://www.youtube.com/watch?v=Vhq4uQI2IYI&NR=1>

<http://www.vectorvest.com/freemovies/demo/vectorvestproducttour/vectorvestproducttour.html>

VectorVest U.S. Welcome back, Dr. Kaner

UniSearch Tool

File • Find • Graph • History • Details • Print • Layout • Quick Test • Add To WatchList • Analysis • New

Date: 9/4/2009 Run Search Return: 15 Return All

Sorted by: VST DESC, Symbol ASC

Searches

New Save Edit Delete

View as: Groups Alphabetical

Searches

- Bear Market Beaters
 - Bear Market Burner
 - Bear market burner no etf
 - BeeLine Boppers II
 - BeeLine Boppers III
 - Best New Highs/BMB
 - Bill's Bear Beaters
 - Bill's Bear Beaters II
 - Kilian's Bear Killer
 - Loewenthal's Express
 - Marathon
 - Richard's Rangers/BMB
 - Seattle Bear Beaters
 - Sector Magic
 - Stalwarts
 - Up, Up and Away
- Cherry Picking
- Delta Searches - Industries
- Delta Searches - Sectors
- Delta Searches - Stocks
- ETFs
- Hi-Lo Searches - Industries
- Hi-Lo Searches - Sectors
- Hi-Lo Searches - Stocks
- kaner
- My Long Searches
- My Short Searches
- ProTrader
- Sector Studies
- Strategies - Aggressive
- Strategies - Bottom Fishing
- Strategies - Candlesticks
- Strategies - Conservative
- Strategies - Price-Volume
- Strategies - Prudent
- Strategies - Short
- Strategies - Special Searches

Description

This strategy finds stocks hitting new 26 week highs, with prices between 2 and 6 dollars and Average Volume greater than

Enter Your Search Criteria

Bear Market Burner

Return Stocks Industries Business Sectors

Selected Date	Parameter	Operator	Value
Date of Search	Stock Price - (Actual)	>	2
Date of Search	Stock Price - (Actual)	<	6
Date of Search	Stock AvgVol - (50 day movir	>	100000
Date of Search	Stock REC - (Recommendatio	<	S
Date of Search	Stock Price - (Split Adjusted)	New High	over 26 Weeks run week
Date of Search	Stock Industry Group	<	Selected Industries
Date of Search	Stock Exchange	<	P

VectorVest Strategies

There are about 250 of these, tailored for different expectations about market performance.

er

17

VectorVest U.S. Stock Analysis and Portfolio Management System

Quick Links: WELCOME

Quick Lookup

Market Analysis: The Color Guard is Neutral

VectorVest at a Glance

Updated as of 9/4/2009 for 8,013 stocks

DATE	CGPRC	CGRT	CGBR
9/4/2009	Y	Y	Y
9/3/2009	Y	Y	Y
9/2/2009	Y	Y	Y
9/1/2009	Y	Y	Y
8/31/2009	Y	Y	Y
8/28/2009	Y	Y	Y

Market Timing Indicator

From 3/4/2009 to 9/4/2009

Strategy of the Week

Leveraged ETFs

Is Using Leveraged ETFs Worth The Risk?

Top Stocks, Industries and Sectors

Top VST Stocks	Top RT Industries	Top RT Sectors
SimpleTech Inc - STET 1.60	Home (Misc Wares) 1.73	Paper 1.54
Bank (Jos A) JOSB 1.58	Retail (Dptmnt Stores) 1.66	Office 1.41
Medfast Inc MED 1.56	Paper 1.54	Instruments 1.31
Broadpoint Sec BPSG 1.50	Chemical (Plastics) 1.53	Insurance 1.28
Ebix Inc EBIX 1.50	Retail (Mail Order/Direct) 1.51	Container 1.26

Strategy

The Price of the VectorVest Composite fell \$0.19 per share since last Friday, changing the Primary wave from Up to Dn. It would have to go down again next week to give a preliminary signal of a sustainable downtrend. Since the Price of the VVC closed at \$21.67 per share on Monday, August 31st, and this price is below today's close of \$21.72 per share, it is quite possible

Current Events

Attend the Newark 2-Day Seminar

You will learn how to make money in this market

SEPTEMBER 11 - 12, 2009

REGISTER TODAY! Click Here OR CALL 1-800-231-0110

Strategy of the Week!

Suppose we sell a trading system with 250 strategies, and analyze the population every week to see which ones performed well.

What if all 250 were no better than a Motley-Fool dartboard? How many should perform statistically significantly better than the market?


(Answer -- 12.5, at the $p \leq .05$ level of significance)

Hmm...
What should
we call the
recommendations
that **didn't** make
money?

*Maybe we
should do
some
research*

THIS IS WHAT I CALL RealMoney

SUBSCRIBE NOW – ONLY \$129.95/yr – \$100 SAVINGS



Where can you find the opportunity for returns like these? In **RealMoney** — my premier information service for investors at all levels of experience.

My personally-assembled team of over 70 Wall Street pros offer **RealMoney** subscribers a wealth of investment wisdom that includes ...

- » A Wall Street-to-you pipeline of fast-breaking news ... market activity ... wealth-building tips ... and portfolio strategies
- » Trade recommendations from our team of experts
- » Up to 15 exclusive reports daily by renowned experts like top chartist Helene Meisler, options expert Jud Pyle, ETF trading pro James "Rev Shark" DePorre, deep-value investing specialist Tim Melvin, and many more Wall Street pros
- » Bulletins twice every trading day alerting you to eye-opening market activity and top news headlines

Subscribe to RealMoney and save \$100 off the price of annual subscription! Take advantage of this great savings so we can team up for **RealMoney** opportunities starting today!

≈ **16% RETURN**
IN JUST OVER 6 DAYS!

John Reese recommended Garmin (GRMN) Sept. 3 when the stock was at \$31.62. It closed Sept. 14 at \$36.81, a 16% gain in a little more than six trading days.

≈ **50% RETURN**
IN LESS THAN A WEEK!!

Jim Cramer recommended SanDisk (SNDK) on Sept. 9 when the stock was trading at \$18.76. It closed Monday at \$20.38, an 8.6% gain in four trading days.

≈ **22% RETURN**
IN 8 DAYS!

Kristen Koh Goldstein bought Abbott (ABT) \$45 calls on Sept. 9 at \$2 apiece; she sold them (announcing the sale) on Sept. 14 at \$3 apiece, for a 50% gain in less than a week.



“COLOR GUARD” on the front page seems to be VectorVest's most unique and important feature. This box speaks to the overall timing of the market:

- VVC Price is the average price of the vector vest composite (the 8013 stocks in the VV database). They use it as an index, like S&P or Dow Jones
- VVC RT is the “relative timing” of the market. The market is on a rising trend for RT > 1 and a declining trend for RT < 1. Based on its published formula (ratios of random variables), I would expect this to have an odd probability distribution.

Market Analysis

Current as of Friday, July 18, 2008



The Color Guard is Neutral

VectorVest advocates caution when buying stocks at this time.

[Click here for details...](#)

VectorVest at a Glance

Updated as of 7/18/2008
for 8,369 stocks

Buys	Sells	Holds
810 (9.7%)	3,939 (47.1%)	3,620 (43.3%)

VVC Price: \$25.94 BSR: 0.21

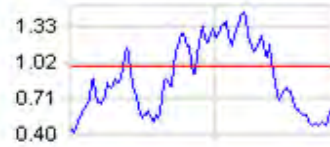
VVC RT: 0.88 MTI: 0.62

VectorVest Color Guard

DATE	CGPRC	CGRT	CGBR
7/18/2008	Y	Y	Y
7/17/2008	G	Y	Y
7/16/2008	Y	Y	Y
7/15/2008	R	R	R
7/14/2008	R	R	Y
7/11/2008	R	Y	Y

Market Timing Indicator

From 1/18/2008 to 7/18/2008



- BSR is the ratio of the number of stocks rated buy to the number of stocks rated sell in the VV database – ignoring the number rated hold. Suppose VV puts “hold” ratings on 8008 stocks, a Buy rating on 4 and a sell rating on 1, I would think this is a flat market, but with a 4-to-1 ratio of buys to sells (ignore the 8008 holds), BSR would have a value of 4.0, a seemingly huge value.
- MTI is the overall Market Timing Indicator and the is described in VectorVest tutorials as a key predictor in their system.

Market Analysis

Current as of Friday, July 18, 2008



The Color Guard is Neutral

VectorVest advocates caution when buying stocks at this time.

[Click here for details...](#)

VectorVest at a Glance

Updated as of 7/18/2008
for 8,369 stocks

Buys	Sells	Holds
810 (9.7%)	3,939 (47.1%)	3,620 (43.3%)

VVC Price: \$25.94 BSR: 0.21

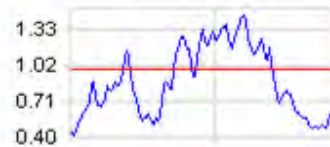
VVC RT: 0.88 MTI: 0.62

VectorVest Color Guard

DATE	CGPRC	CGRT	CGBR
7/18/2008	Y	Y	Y
7/17/2008	G	Y	Y
7/16/2008	Y	Y	Y
7/15/2008	R	R	R
7/14/2008	R	R	Y
7/11/2008	R	Y	Y

Market Timing Indicator

From 1/18/2008 to 7/18/2008



- The color system is summary of trends over the past few days / few weeks.
- According to VectorVest:
 - yellows mean there is no trend to follow,
 - red means the market is declining and you should NOT buy anything tomorrow, and
 - green means the market is rising and you feel good about buying.

VectorVest in their own words

Dr. DiLiddo on VectorVest - THE COLOR GUARD, CLARIFIED - Windows Internet Explorer

http://www.vectorvest.com/BLOG/post/2008/09/12/THE-COLOR-GUARD-CLARIFIED.aspx

vectorvest colorguard

THE COLOR GUARD, CLARIFIED
By Dr. Bart DiLiddo Friday, 09/12/2008

I have received some feedback recently that the Color Guard is confusing and hard to understand, so this essay is dedicated to explaining, as well as I can, how it works.

Think of the Color Guard as you would a traffic light. Green means go, it's OK to buy stocks. Yellow means caution, it may or may not be OK to buy stocks. Red means stop, don't buy any stocks.

Can it really be that simple? In a perfect world, the answer would be yes, but there's more here than meets the eye. For example, I'm not going to go through a green traffic light if I see a truck in my path, running a red light. Even though I have the green, I always look both ways to make sure it's safe to proceed. This is why our trend indicators, the Primary Wave and the Market Timing Indicator, are important parts of the Color Guard.

We use colors to report the up and down movements of the Price of the VectorVest Composite, the Relative Timing of the V V C and the Buy to Sell Ratio, and we use Up and Dn signals to report the direction of our Trend Indicators. Put it all together and we have a set of three lights and two indicators. Here's a summary of the most likely combinations of light colors and trends, what they mean and the suggested action to take:

LIGHTS.....	TRENDS	MEANING.....	ACTION
3 Green.....	UpUp	The Color Guard is Bullish.....	Buy...
2 Green.....	UpUp	The Color Guard is Somewhat Bullish	Buy...
1 Green.....	UpUp	The Color Guard is Mildly Bullish..	Buy...
1 Green.....	UpDn	The Color Guard is Mildly Bullish..	Buy w Caution.

Search
Enter search term Search
☐ Include comments in search
[Subscribe](#)

Page List
[About Dr. Bart A. DiLiddo](#)
[Bottom Fishing](#)
[Guidance On Strategy And Market Timing](#)
[How To Protect Profits](#)
[How To Protect Profits](#)
[How To Short Stocks](#)
[The Color Guard, Clarified](#)
[The Daily Color Guard](#)
[Timing The Market](#)
[Use Options To Supercharge Profits](#)

RecentPosts
RETIREMENT STRATEGY - PART I
Comments: 0
Rating: 5 / 1
PRICELESS

Exploratory Software Test Automation

Copyright © 2009 Cem Kaner

23

VectorVest in their own words

3 Yellow....	UpUp	The Color Guard is Neutral.....	Buy w Caution.
3 Yellow....	UpDn	The Color Guard is Neutral.....	Buy w Caution.
3 Yellow....	DnUp	The Color Guard is Neutral.....	Do Not Buy.
3 Yellow....	DnDn	The Color Guard is Neutral.....	Do Not Buy.
1 Red.....	DnUp	The Color Guard is Mildly Bearish..	Do Not Buy.
1 Red.....	DnDn	The Color Guard is Mildly Bearish..	Do Not Buy.
2 Red.....	DnDn	The Color Guard is Somewhat Bearish	Do Not Buy.
3 Red.....	DnDn	The Color Guard is Bearish.....	Do Not Buy.

Given this array of light combinations, trends, meanings and actions, I can see why some people may be confused by the Color Guard. But here's how to make it real simple: IT'S ALWAYS OK TO BUY STOCKS WHEN THE PRIMARY WAVE IS UP.

The Primary Wave, shown on the left side of the Trends column, is discussed at some length in my essay of July 9, 2004. The Underlying Trend, shown on the right side of the Trend column, is discussed in my essay of July 23, 2004. A comprehensive review of our Market Timing System was presented in a series of seven essays from July 9, 2004 to August 20, 2004.

Comments: 0
Rating: 3 / 2
BUYING LOW AND SELLING HIGH
Comments: 0
Rating: 5 / 1
ON THE ROAD AGAIN
Comments: 0
Rating: 5 / 1
BECOME A GREEN LIGHT BUYER
Comments: 6
Rating: 3.4 / 19
VECTORVEST OPTION TOOLS
Comments: 0
Rating: 0 / 0
ONE DAY AT A TIME
Comments: 1
Rating: 3.7 / 3
THE YELLOW BRICK ROAD PORTFOLIO
Comments: 0
Rating: 5 / 1
MANAGING THE YELLOW BRICK ROAD PORTFOLIO
Comments: 0

from ... **THE COLOR GUARD, CLARIFIED**
Bart DiLiddo (VectorVest founder)

Attend the Newark 2-Day Seminar
You will learn how to make money in this market
SEPTEMBER 11 - 12, 2009
Sheraton Newark Airport Hotel, 120 Frontage Rd, Newark, NJ 07114
REGISTER TODAY! Click Here OR CALL 1-800-231-0110
Only \$195.00 Expires 8/4/09

Exploratory Software Test Automation

Copyright © 2009 Cem Kaner

24

A little research

To study the ColorGuard system as a predictor of the market, I downloaded Standard and Poors' S&P-500 index prices from January 4, 1999* through early Sept 2009.

I then computed percentage price changes:

- percent gain or loss in the S&P compared to the current day
- percent gain or loss between the current day value and the value 5 trading days from the current day.
- after 15 trading days
- after 30 trading days.

I also looked at 2-day, 3-day and 4-day for some analyses, but the results were the same as 1-day and 5-day so I stopped bothering.

- The average day-to-day change in the market was 0.0027% (flat over 10 years)

* Available ColorGuard data appeared to start in late 1998

Results

1. Correlation between **RT** (relative timing) and future S&P index price was slightly **negative** (**-0.034** for next-day price, **-0.038** for 5-day, **-0.019** for 15-day and **0.010** for 30-day).
 - The effect trends toward zero as you go further in the future (as a predictor of 30-days in the future). With over 2600 days of data, these tiny correlations are probably statistically significant, but it's a tiny effect.
2. Correlation between **Buy Sell ratio** and future price is **-0.012** for next-day, **-0.011** for 5 day, **0.018** for 15 day and **0.032** for 30 day.
 - Like relative timing, as an indicator for short term trading decisions, this is at best, worthless.
3. Correlation between **MTI** and future price is **-0.025** for next day, **-0.025** for 5-day (the same number is not a typo), **-0.004** for 15 day and **-0.016** for 30-day.

*When these indicators predict "going up",
the market on average goes slightly down.*

A Few More Results

From 1999 to September 2009:

- 289 trading days rated **GREEN** / **GREEN** / **GREEN** (**GGG**) (**buy**)
- 285 rated **RED** / **RED** / **RED** (**don't buy**).
- After **GGG** days, S&P index went **DOWN** an average of
 - **-0.05%** the day after a **GGG** rating,
 - **-0.24%** five days after,
 - **-0.14%** 15 days after, and
 - **-0.33%** 30 days after.
- After **RRR** days, S&P went **UP** an average of
 - **0.24%** the day after an **RRR**,
 - **0.29%** five days after,
 - **0.42%** 15 days after and
 - **1.08%** in the 30 days after an **RRR**

Basic concepts

- Let's define a "strategy" as a combination of decision rules:
 - what to buy
 - how much to buy
 - when to buy it
 - when to sell it
 - how much to sell
- Many strategies require a human decision-maker applying human judgment.
- Whether by person or machine, all decisions are made under uncertainty

Simplistic strategies

- Buy in the early morning (market open), sell at market close (in a generally rising market)
 - (How do you know if you're in a generally rising market?)
- Buy at close, sell the next morning (in a generally declining market)
- Buy a stock when it drops more than D% in one day and sell it back when the stock regains $\frac{1}{2}$ of its loss
- Buy a stock when it hits a 52-week high and sell (a) after B% (e.g. 25%) rise or 10% trailing stop

These are probably all losers, but how can we tell?

To Develop a Strategy, Build a Model

1. start with a plausible hypothesis
 - in practice, this is the hardest step and the one that requires the most investigation
2. decide what data to test it on
3. get the data
4. what's the right test?
5. if the model proves itself wrong
 - study the fine grain of the data for evolution to next model
6. if model appears right
 - what replications are needed, on what data, to check this further?

1. Start with a plausible hypothesis

- in practice, this is the hardest step and the one that requires the most investigation
- For now, let's select this heuristic:

*In a generally rising period, buy at
market open, sell at market close*

(reflects market optimism)

2. Decide what data to test it on

- Should we test on individual stocks or an aggregate?
 - If individual, should we sample from a specific pool (e.g. Wireless Internet stocks (think iPhone) might behave differently from consumer stocks like Taco Bell)?
 - How should we select from the pool?
- What time interval should we test on?
 - generally rising?
 - generally falling?
 - based on indicators (like consumer confidence)?
 - random?

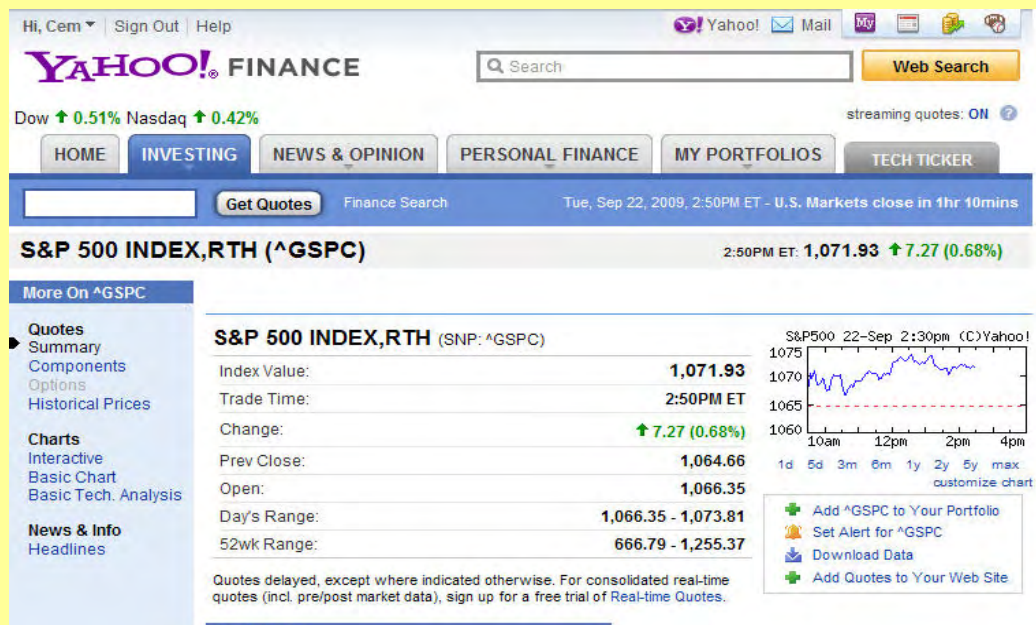
2. Decide what data to test it on

- To simplify our example, let's choose the Standard & Poors index
- from March 9 to present



3. Get the data

- In this case, getting the data is easy.
- Yahoo has it.



3. Get the data

- But how do we know that it's right?
- (This is a serious problem across different data sources. Examples in oral lecture.)
- HOW DO WE **GET** THE DATA?

The screenshot shows the Yahoo! Finance website interface. At the top, there's a navigation bar with 'HOME', 'INVESTING', 'NEWS & OPINION', 'PERSONAL FINANCE', 'MY PORTFOLIOS', and 'TECH TICKER'. Below this, a search bar and 'GET QUOTES' button are visible. The main content area displays the 'S&P 500 INDEX, RTH (^GSPC)' with a current price of 1,072.23 and a green upward arrow indicating a 0.71% increase. To the left, a sidebar lists various options like 'Quotes', 'Summary', 'Components', 'Options', 'Historical Prices', 'Charts', and 'News & Info'. The 'Historical Prices' section is active, showing a 'SET DATE RANGE' form with 'Start Date' as Mar 9, 2009, and 'End Date' as Sep 22, 2009. Below the form, a table of historical prices is displayed.

Date	Open	High	Low	Close	Volume	Adj Close*
21-Sep-09	1,067.14	1,067.28	1,057.46	1,064.66	4,615,280,000	1,064.66
18-Sep-09	1,066.60	1,071.52	1,064.27	1,068.30	5,607,970,000	1,068.30
17-Sep-09	1,067.87	1,074.77	1,061.20	1,065.49	6,668,110,000	1,065.49
16-Sep-09	1,053.99	1,068.76	1,052.87	1,068.76	6,793,529,600	1,068.76

4. What's the right test?

- My test is to calculate
 $\text{delta} = \text{Closing index value} - \text{Opening index value}$
 for every day in the time period
 and then calculate average delta per day

	A	B	C	D	E	F	G	H	I	J
1	Date	Open	High	Low	Close	Volume	Adj Close	CLOSE - OPEN	TOTAL	AVERAGE GAIN
2	9/21/2009	1067.14	1067.28	1057.46	1064.66	4.62E+09	1064.66	-2.48	407.27	2.97
3	9/18/2009	1066.6	1071.52	1064.27	1068.3	5.61E+09	1068.3	1.7		
4	9/17/2009	1067.87	1074.77	1061.2	1065.49	6.67E+09	1065.49	-2.38		
5	9/16/2009	1053.99	1068.76	1052.87	1068.76	6.79E+09	1068.76	14.77		
6	9/15/2009	1049.03	1056.04	1043.42	1052.63	6.19E+09	1052.63	3.6		
7	9/14/2009	1040.15	1049.74	1035	1049.34	4.98E+09	1049.34	9.19		
8	9/11/2009	1043.92	1048.18	1038.4	1042.73	4.92E+09	1042.73	-1.19		
9	9/10/2009	1032.99	1044.14	1028.04	1044.14	5.19E+09	1044.14	11.15		
10	9/9/2009	1025.36	1036.34	1023.97	1033.37	5.2E+09	1033.37	8.01		

WOO
HOO!!

5. If the model proves itself wrong

That didn't happen.

So we don't have to worry about it.

(yet)

6. If the model appears right

What replications are needed, on what data, to check this further?

- Replications on rising markets in previous years?
- Replications on falling markets?
- Replications on broader market (S&P is a 500-stock subset of 15,500 stock market)
- Replication across geographic segments (Chinese stocks? Israeli stocks? UK stocks?) (Do these add noise that should be chopped from our buying strategy?)

CAUTION: From my limited study, 2009 appears anomalous.

As far as I can tell, betting on this in the future is as likely to yield Boo Hoo as Woo Hoo.

1. But maybe we get a new hypothesis

Can we strengthen the predictable-rise-during-the-day hypothesis?

- study the fine grain of the data
- decide whether the hypothesis is wrong or incomplete
- if incomplete:
 - vary conditions as (potentially) appropriate
 - If the underlying theory is a daily rise due to optimism
 - » should we buy only when Consumer Confidence is up (we do have historical data)?
 - » should we focus on stocks recently upgraded by analysts?
 - » what else could enhance a general optimism, increasing its impact for a specific stock or industry or sector?
 - » What if we tried EVERY variable?

Let's illustrate this with an example

Royal Bank of Scotland Preferred Series T

Preferred Stocks:

- Owned by shareholders (like common stock), but usually nonvoting
- Stock pays dividends at a (typically) fixed rate, comparable to paying interest on a debt
- This is like debt except that the company doesn't have to pay it back
- Dividends can be mandatory or discretionary
- RBS-Preferred Series T (RBS+T or RBSPT)
 - discretionary, \$1.812 per year (\$0.453 per quarter)
 - sold for \$25
 - at \$25, \$1.812 is 7.25% interest
 - at \$11.99, 1.812 is 15.11%
 - (a HUGE dividend if you think RBS is a survivor)

A candlestick chart for RBS/T



- Prices on the top
- Volume on the bottom
- Shows:
 - High price for the day
 - Starting price
 - Ending price
 - Low price for the day
 - Bar is red if Start < End
- This chart shows:
 - Royal Bank of Scotland Preferred stock, Series T
- On August 20, 2009,
 - Opened at \$15.01
 - High at \$15.29
 - Low at \$11.27
 - Closed at \$11.99
- On September 25
 - Closed at \$14.02
- Wouldn't it be nice to
 - Buy at \$11.99
 - and sell at \$14.02?

Some big changes...



The big drop on 8/20/09

Aug 20 - Fitch Ratings has downgraded the ratings of hybrid securities at Lloyds Banking Group plc (LBS), Royal Bank of Scotland Group plc (RBS), ING Group, Dexia Group, ABN Amro, SNS Bank, Fortis Bank Nederland and BPCE and certain related entities. The downgrade reflects increased risk of deferral of interest payments after the European Commission (the "Commission") clarified its stance on bank hybrid capital, and in particular the application of the concept of "burden-sharing". A full list of ratings actions is available at the end of this commentary.

The Commission's recent statements confirm Fitch's view that government support for banks may not extend to holders of subordinated bank capital (see 4 February 2009 comment "Fitch Sees Elevated Risk of Bank Hybrid Coupon Deferral in 2009" on www.fitchratings.com). Fitch has already taken significant rating actions on the hybrid capital instruments of ailing banks within the EU and elsewhere. Nevertheless, in the light of the latest Commission statements, Fitch is applying additional guidelines in its ratings of hybrid capital instruments issued by EU financial institutions. These are outlined in a report published today, entitled "Burden Sharing and Bank Hybrid Capital within the EU." A second report; "UK Banks and State Aid: 'A Burden Shared'", which is also published today, discusses the implications for bondholders of UK banks that have received state aid.

In particular, Fitch would highlight that a bank that has received state aid and is subject to a name-specific restructuring process will likely have a hybrid capital rating in the 'BB' range or below, with most ratings on Rating Watch Negative (RWN), indicating the possibility of further downgrades. Banks which Fitch believes are subject to significant state aid beyond broad-based confidence building measures will likely have a hybrid capital rating in the 'B' range or below, and be on RWN. Fitch will apply these guidelines to banks where a formal state aid process has not yet been established, but where Fitch believes such a process is likely to arise. ...

The securities affected are as follows: The Royal Bank of Scotland Group plc -- Preferred stock downgraded to 'B' from 'BB-' and remains on RWN (and a bunch of other banks)

Information disparity



Exploratory Software Test Automation

Copyright © 2009 Cem Kaner

45

Information disparity (RBS 9/4/09)



Interactive Brokers
The Professional's Gateway to the World's Markets

Message Notification

Sep 04, 2009

You have 1 NORMAL priority message(s) for your account UXXX176.

Priority: NORMAL

1 RBS PRR@NYSE Corp Action Notification

RBS PRR@NYSE (Royal Bank of Scotland Group PLC) announced a cash dividend with ex-dividend date of 20090911. The declared cash rate is USD 0.382825.

The above (i) was compiled by IB on a best efforts basis from information IB received from third party vendors; (ii) may contain errors or omissions; and (iii) is subject to change without prior or additional notice. IB does not warrant that the above is accurate, timely, or complete. The above is intended for only IB Clients, and it does not constitute a recommendation or advice by IB, and IB Clients may not rely upon it. IB Clients are urged to verify the information prior to using it in their investing and trading decisions, including through reference to independent financial news resources.

Sep 04, 2009 04:01 EST

Interactive Brokers LLC, member [NYSE](#), [FINRA](#), [SIPC](#)

I got this note from IB at 4 am on 9/4. This news didn't show up on the RBS site or at several brokerage or news sites until end of day or next day (or later).

So, some people were buying on the news that dividends are coming. Others selling because they thought dividends were not coming.

From a modeling perspective, we are working with nonstationary mixture distributions and underlying distributions that appear to have thick tails (many outlying values) and are often asymmetric. Research has to be intensely empirical (history & simulations) because the theoretical math is so difficult.

Exploratory Software Test Automation

Copyright © 2009 Cem Kaner

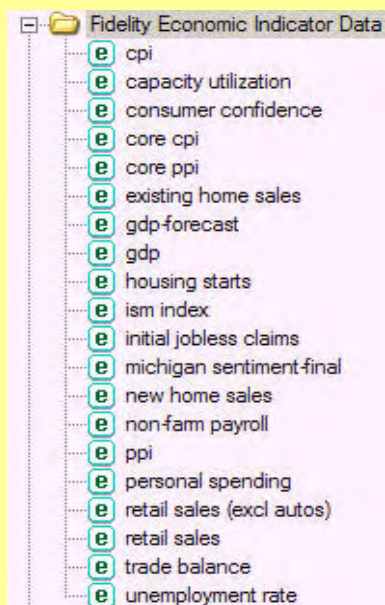
46

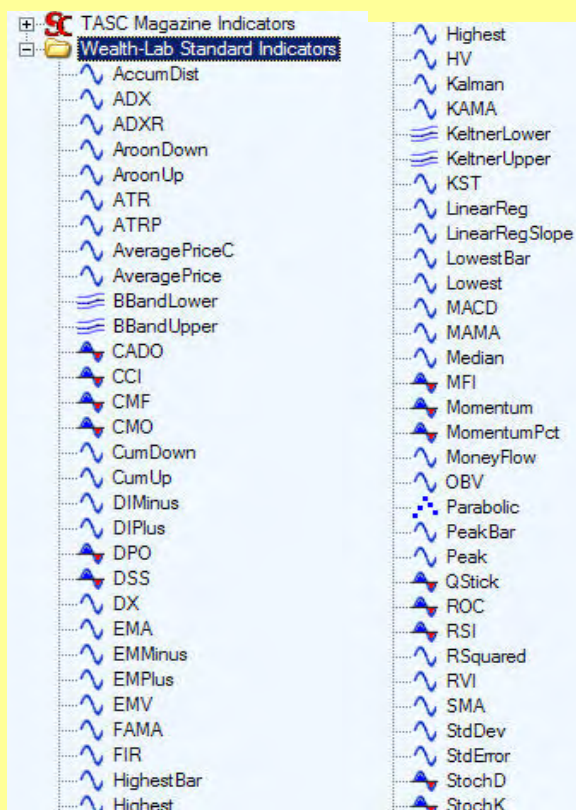
So how do we model this?

- My preference is to look at the fine grain of the data, for interactions among several variables that :
 - haven't been studied well enough together (combine "fundamental" and "technical" variables)
 - are predictive under a narrower set of circumstances than all stocks (or all stocks in a sector) or at all times
- We have hundreds of potentially available variables to work with.
- Their values are all probabilistic
- Some variables will be better predictors than others.

Variables Available in Fidelity WealthLab:

You can find values for almost any point in time





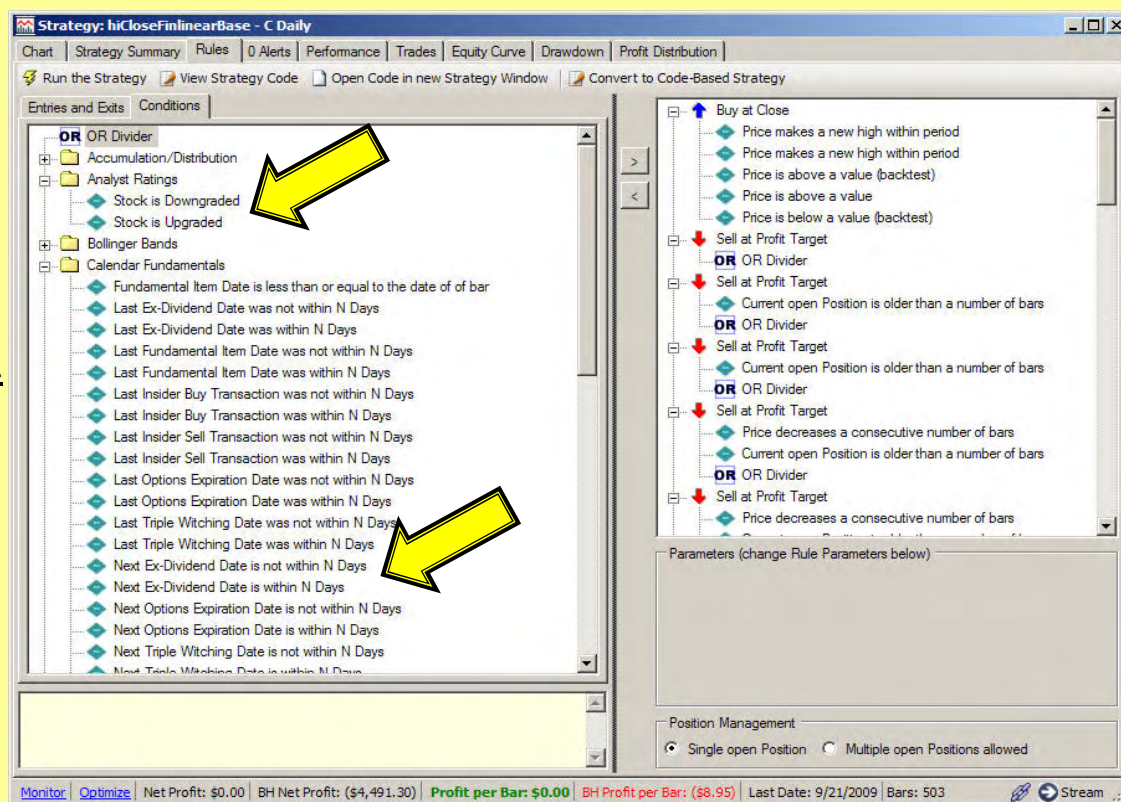
Variables

You can calculate any delta over time or any delta of any ratio of these over time

Which should we look at?



Even
more
Wealth-
Lab data



Screening for predictive quality

Screening for predictive relationships is just screening, not proving.

If we want a heuristic good enough to bet on, we need

- extensive replication,
- discovery of conditions under which the variable (or strategy) is ineffective, and
- a rationale that makes sense that allows us to make (and test) predictions about when the strategy will be effective and when it will not.

What does this
investment example
show us?

First, it IS testing

Imagine these roles:

- individual investor trying to avoid making terrible mistakes and losing all her retirement money
- technical expert, hired by Securities & Exchange Commission to help them investigate stock-market related fraud
- analyst helping an investment firm assess effectiveness of investment strategies
- individual investor trying to assess effectiveness of investment-guidance tools **that he is writing**
- analyst working at an investment firm that creates/sells investment-guidance tools:
 - writing the code
 - evaluating the usability and basic dependability of the product
 - evaluating the effectiveness of the product

These are
all testing
roles

This kind of testing can pay pretty well, too:

Goldman Sachs compensation: \$1 mn per employee in sight

4 Jul 2009, 2105 hrs IST, REUTERS,

<http://economictimes.indiatimes.com/articleshow/4777281.cms?prtpage=1>

NEW YORK: The average Goldman Sachs Group Inc employee is within striking distance of \$1 million in compensation and benefits this year, just nine months after the bank received a \$10 billion US government bailout.

The figure will likely fuel criticism of the politically connected bank, especially amid the widening recession and rising unemployment. In addition to the bailout, Wall Street's biggest surviving securities firm also benefited from several other government schemes during the depths of last year's financial crisis.

Goldman on Tuesday said money set aside for pay surged 75 percent in the second quarter. Compensation and benefits costs were \$6.65 billion, up 47 percent from the equivalent quarter in 2008.

Given a 16 percent reduction in staff from last year, to 29,400, the bank set aside an average \$226,156 per employee in the second quarter, up from \$129,200 in a year ago. If the quarterly figure is annualized, it comes to **\$904,624 per employee.**

A Critical Distinction

- People who create (or use) (or improve) technology to model the market are called "QUANTS."
 - THEY ARE CENTRAL TO THE BUSINESS
 - THE BUSINESS BASES ITS DECISIONS ON THEIR WORK
- People who write (or test) code are called "technicians"
 - Even if they are doing implementation for the quants
 - Their focus is on the generic tasks of development
 - Instead of the specialized tasks of the business
 - They are cheap, expendable, and outsourceable (but certifiable)

Testing focused on business value

- When we study "computing" as a general field (or software testing) (or software engineering) we often abstract away the underlying complexities of the subject matter we are working in.
- A computer program is not just "a set of instructions for a computer." It is an attempt to help someone do something. The program:
 - makes new things possible, or
 - makes old things easier, or
 - helps us gain new insights, or
 - brings us new experiences (e.g. entertainment)
- Programs provide value to companies
 - Some programs tie directly to the core value-generating or risk-mitigating activities in the company
 - Especially in organizations that see computing as a technology rather than as a goal in itself, your value to the organization rises if your work actively supports the business value of the software.

Testing focused on business value

- Every evaluative step in this example involved testing
- This work requires extensive technical knowledge, but
 - "technical" = the conceptually difficult areas in the business domain
 - "technical" \neq programming
- For years, the highest paid testers on Adobe Illustrator were graphic artists
- For several years at WordStar, the most senior tester was a printer expert
- Consider what testing you could do if you deeply understood:
 - actuarial mathematics (insurance risk modeling)
 - the technology (e.g. data mining) and psychology of customer relationship management and sales prospecting
 - taxation systems (e.g. as applied in employee compensation software)
 - airline reservation systems
 - the human factors of creative work (draw / video / text / music)

Along with being focused on business value

This testing was heavily automated

But there was no hint of regression testing

Typical Testing Tasks

Analyze product & its risks

- benefits & features
- risks in use
- market expectations
- interaction with external S/W
- diversity / stability of platforms
- extent of prior testing
- assess source code

Develop testing strategy

- pick key techniques
- prioritize testing foci

Design tests

- select key test ideas
- create tests for each idea

Run test first time (often by hand)

Evaluate results

- Troubleshoot failures
- Report failures

Manage test environment

- set up test lab
- select / use hardware/software configurations
- manage test tools

Keep archival records

- what tests have we run
- trace tests back to specs

If we create regression tests:

- Capture or code steps once test passes
- Save “good” result
- Document test / file
- Execute the test
 - Evaluate result
 - Report failure or
 - Maintain test case

This contrasts the variety of tasks commonly done in testing with the narrow reach of UI-level regression automation. This list is illustrative, not exhaustive.

GUI-Level Regression Testing: Commodity-Level Test Automation

- addresses a narrow subset of the universe of testing tasks
- re-use existing tests
 - these tests have one thing in common: the program has passed all of them
 - little new information about the product under test
 - rarely revised to become harsher as the product gets more stable, so the suite is either too harsh for early testing or too simplistic / unrealistic for later testing
 - often address issues (e.g. boundary tests) cheaper and better tested at unit level
- underestimate costs of maintenance and documentation
 - capture/replace costs are exorbitant. Frameworks for reducing GUI regression maintenance costs are implementable, but they require development effort and the maintenance-of-tests costs are still significant
 - test documentation is needed for "large" (1000+) test suites or no one will remember what is actually tested (and what is not). Creating / maintaining these docs is not cheap
- project inertia: refers to the economic resistance to any improvement to the code that would require test maintenance and test doc maintenance.

The Underlying Tasks of Test Automation

- **Theory of error**
What kinds of errors do we hope to expose?
- **Input data**
How will we select and generate input data and conditions?
- **Sequential dependence**
Should tests be independent? If not, what info should persist or drive sequence from test N to N+1?
- **Execution**
How well are test suites run, especially in case of individual test failures?
- **Output data**
Observe which outputs, and what dimensions of them?
- **Comparison data**
If detection is via comparison to oracle data, where do we get the data?
- **Detection**
What heuristics/rules tell us there **might** be a problem?
- **Evaluation**
How **decide** whether X is a problem or not?
- **Troubleshooting support**
Failure triggers what further data collection?
- **Notification**
How/when is failure reported?
- **Retention**
In general, what data do we keep?
- **Maintenance**
How are tests / suites updated / replaced?
- **Relevant contexts**
Under what circumstances is this approach relevant/desirable?

Theory of Error

- Common programming problems that would be caught in good unit testing
- Failure to conform to a specification
- Failure to enable a desirable benefit
- Computational errors
 - obvious
 - boundary cases
 - subtle
- Communications problems
 - protocol error
 - their-fault interoperability failure
- Resource unavailability or corruption, driven by
 - history of operations
 - competition for the resource
- Race conditions or other time-related or thread-related errors
- Failure caused by toxic data value combinations
 - that span a large portion or a small portion of the data space
 - that are likely or unlikely to be visible in "obvious" tests based on customer usage or common heuristics

The Telenova Station Set

1984. First phone on the market with an LCD display.

One of the first PBX's with integrated voice and data.

108 voice features, 110 data features. accessible through the station set



The Telenova stack failure

```
July 4, 1985      12:01 PM  Ext: 257  
Directory Admin  Messages Voice Data
```

```
1-(212)662-7777 Connected  Ext: 567  
Transfer Record  Conference Park Acct
```

```
Please enter selection  
LvlMss GetMss Greeting Code
```

```
Ted K. waiting      Wt:1 Hd:0  
I'llCall CallLater PlsWait Answ
```

```
Select a call & lift handset Wt:5 Hd:5  
Ted K. Peter T. Trunk 6 Trk 2Trk 7
```

```
Kenix 3 Connected for Data  
Transfer Baud EndCall Park Acct
```



Context-sensitive display
10-deep hold queue
10-deep wait queue

Stack Failure

- System allowed up to 10 calls on hold
- Stored call-related data on a held-call stack
- Under a rare circumstance, held call could be terminated but the stack entry not cleared
 - If the number of
 - calls actually on hold, plus
 - not-cleared terminated calls
 - exceeded 20
 - the phone rebooted due to stack overflow
- In testing in the lab:
 - this bug never showed up even though testing achieved 100% statement and branch coverage in the relevant parts of the code (stack cleanup methods masked the error and usually avoided failure in the field from this bug)
- In the field, this required a long sequence of calls to a continuously-active phone.
- Failure was irreproducible unless you considered the last 1-3 hours of activity

Telenova stack failure

Having found and fixed
the hold-stack bug,
should we assume
that we've taken care of the problem
or that if there is one long-sequence bug,
there will be more?

Hmmm...
If you kill a giant cockroach in your kitchen,
do you
assume you've killed the last bug?
or do you
call the exterminator?

Simulator with probes

Telenova (*) created a simulator

- generated long chains of random events, emulating input to the system's 100 phones
- could be biased, to generate more holds, more forwards, more conferences, etc.

Programmers added probes (non-crashing asserts that sent alerts to a printed log) selectively

- can't probe everything b/c of timing impact
-

(*) By the time this was implemented, I had joined Electronic Arts. This summarizes what colleagues told me, not what I personally witnessed.

Simulator with probes

- After each run, programmers and testers tried to replicate failures, fix anything that triggered a message. After several runs, the logs ran almost clean.
- At that point, shift focus to next group of features.
- Exposed lots of bugs
- Many of the failures probably corresponded to hard-to-reproduce bugs reported from the field.
 - These types of failures are hard to describe/explain in field reports

Telenova stack failure

- Simplistic approaches to path testing can miss critical defects.
- Critical defects can arise under circumstances that appear (in a test lab) so specialized that you would never intentionally test for them.
- When (in some future course or book) you hear a new methodology for combination testing or path testing:
 - test it against this defect.
 - If you had no suspicion that there was a stack corruption problem in this program, would the new method lead you to find this bug?

A second case study: Long-sequence regression

- Welcome to “Mentsville”, a household-name manufacturer, widely respected for product quality, who chooses to remain anonymous.
- Mentsville applies wide range of tests to their products, including unit-level tests and system-level regression tests.
 - We estimate > 100,000 regression tests in “active” library

Theory of Error

- Common programming problems that would be caught in good unit testing
- Failure to conform to a specification
- Failure to enable a desirable benefit
- Computational errors
 - obvious
 - boundary cases
 - subtle
- Communications problems
 - protocol error
 - their-fault interoperability failure
- Resource unavailability or corruption, driven by
 - history of operations
 - competition for the resource
- Race conditions or other time-related or thread-related errors
- Failure caused by toxic data value combinations
 - that span a large portion or a small portion of the data space
 - that are likely or unlikely to be visible in "obvious" tests based on customer usage or common heuristics

A second case study: Long-sequence regression

- Long-Sequence Regression Testing (LSRT)
 - Tests taken from the pool of tests ***the program has passed in this build.***
 - The tests sampled are run in random order until the software under test fails (e.g crash).
- Note:
 - these tests are no longer testing for the failures they were designed to expose.
 - these tests add ***nothing*** to typical measures of coverage, because the statements, branches and subpaths within these tests were covered the first time these tests were run in this build.

Long-sequence regression testing

- Typical defects found include timing problems, memory corruption (including stack corruption), and memory leaks.
- Recent (2004) release: 293 reported failures exposed 74 distinct bugs, including 14 showstoppers.
- Mentsville's assessment is that *LSRT exposes problems that can't be found in less expensive ways.*
 - troubleshooting these failures can be very difficult and very expensive
 - wouldn't want to use LSRT for basic functional bugs or simple memory leaks--too expensive.

Long-sequence regression testing

- LSRT has gradually become one of the fundamental techniques relied on by Mentsville
 - gates release from one milestone level to the next.
- Think about testing the firmware in your car, instead of the firmware in Mentsville's devices:
 - fuel injectors, brakes, almost everything is computer controlled
 - for how long a sequence would you want to run LSRT's to have confidence that you could drive the car 5000 miles without failure?
- what if your car's RAM was designed to be reset only after 100,000 miles, not when you turn the car off?

Theory of Error

- Common programming problems that would be caught in good unit testing
- Failure to conform to a specification
- Failure to enable a desirable benefit
- Computational errors
 - obvious
 - boundary cases
 - subtle
- Communications problems
 - protocol error
 - their-fault interoperability failure
- Resource unavailability or corruption, driven by
 - history of operations
 - competition for the resource
- Race conditions or other time-related or thread-related errors
- Failure caused by toxic data value combinations
 - that span a large portion or a small portion of the data space
 - that are likely or unlikely to be visible in "obvious" tests based on customer usage or common heuristics

High-Volume Combination Testing

- Lots of academic research
- Instead of combination-test sampling heuristics like all-pairs or domain testing
 - generate large (maybe exhaustive) sets of combination tests
 - e.g. if there are 10 variables with 4 values of interest each, there are 4^{10} possible tests, so with good tools, we could generate each 10-variable combination and run every test
 - the limiting factor is availability of an oracle (how can you tell if the program failed?)

Theory of Error

- Common programming problems that would be caught in good unit testing
- Failure to conform to a specification
- Failure to enable a desirable benefit
- Computational errors
 - obvious
 - boundary cases
 - subtle
- Communications problems
 - protocol error
 - their-fault interoperability failure
- Resource unavailability or corruption, driven by
 - history of operations
 - competition for the resource
- Race conditions or other time-related or thread-related errors
- Failure caused by toxic data value combinations
 - that span a large portion or a small portion of the data space
 - that are likely or unlikely to be visible in "obvious" tests based on customer usage or common heuristics

Function Equivalence Testing

Doug Hoffman worked for MASPAC (the Massively Parallel computer, 64K parallel processors).

The MASPAC computer has several built-in mathematical functions. We're going to consider the Integer square root.

This function takes a 32-bit word as an input. Any bit pattern in that word can be interpreted as an integer whose value is between 0 and $2^{32}-1$. There are 4,294,967,296 possible inputs to this function.

- How many of them should we test?
- How many would you test?
- Hoffman, Exhausting Your Testing Options, at <http://www.softwarequalitymethods.com/H-Papers.html#maspar>

It's a life-critical system...

- To test the 32-bit integer square root function, Hoffman checked all values (all 4,294,967,296 of them). This took the computer about 6 minutes to run the tests and compare the results to an oracle.
- There were 2 (two) errors, neither of them near any boundary. (The underlying error was that a bit was sometimes mis-set, but in most error cases, there was no effect on the final calculated result.) Without an exhaustive test, these errors probably wouldn't have shown up.
- What about the 64-bit integer square root? How could we find the time to run all of these? If we don't run them all, don't we risk missing some bugs?

Ten Examples of High Volume Automated Tests

1. Simulator with probes
2. Long sequence regression testing
3. Function equivalence testing (comparison to a reference function)
4. Comparison to a computational or logical model
5. Comparison to a heuristic prediction, such as prior behavior
6. State-transition testing without a state model (dumb monkeys)
7. State-transition testing using a state model (terminate on failure rather than after achieving some coverage criterion)
8. Functional testing in the presence of background load
9. Hostile data stream testing
10. Random inputs to protocol checkers

See Kaner, Bond, McGee, www.kaner.com/pdfs/highvolCSTER.pdf

The Tasks of Test Automation

- **Theory of error**
What kinds of errors do we hope to expose?
- **Input data**
How will we select and generate input data and conditions?
- **Sequential dependence**
Should tests be independent? If not, what info should persist or drive sequence from test N to N+1?
- **Execution**
How well are test suites run, especially in case of individual test failures?
- **Output data**
Observe which outputs, and what dimensions of them?
- **Comparison data**
IF detection is via comparison to oracle data, where do we get the data?
- **Detection**
What heuristics/rules tell us there **might** be a problem?
- **Evaluation**
How **decide** whether X is a problem or not?
- **Troubleshooting support**
Failure triggers what further data collection?
- **Notification**
How/when is failure reported?
- **Retention**
In general, what data do we keep?
- **Maintenance**
How are tests / suites updated / replaced?
- **Relevant contexts**
Under what circumstances is this approach relevant/desirable?

Input data

How will we select & generate input & and conditions?

Selection of values to test

- single items or combinations of variables
 - managing combinatorial explosion
 - managing data dependencies
- theory of selection
 - cover the data space (via sampling)
 - mitigate hypothesized risks
 - match usage profile(s)
- approach to sampling
 - exhaustive?
 - random sample (plus edges?)
 - interval sample (with edges?)
 - stratified random sample
 - historical actual data

Generation of test inputs

- what dimensions do we generate
 - timing (time between different parts of the input)
 - movements (e.g. mouse gestures when drawing)
 - sequence of subtasks
 - edits
 - final input value
- how do we generate them
 - capture human activity
 - capture activity and then vary parameters algorithmically
 - create the data algorithmically

Sequential dependence:
Should tests be independent?

If not, what info should persist or drive sequence from test N to N+1?

- independence
 - does the automaton enforce this via clean-up algorithms
- flow of control
 - e.g. state-transition testing. Every test is the precursor to state transition to the next.
 - what is the underlying model, how is validated against the software, how maintained, and is failure to reach the next state detected as a bug?
- dirty system
 - Test N does not clean up before Test N+1 (like real use--you don't reboot a printer after each print job)
 - how do we monitor the state of the system after Test N?

Execution:
How well are test suites run,
especially in event of individual test failures?

- What is the test execution framework
 - how do you add / subtract new tests
 - how do you know what you have, and have run
 - simple execution of linear script or a general-purpose interpreter that supports reusable methods
 - response to failure: log / clean-up / continue?

Output data:

Observe which outputs & what dimensions of them?

- Which outputs will we observe?
 - (Remember Hoffman's lesson--every test generates an unmanageably large set of results, so we choose what we look at.)
- Which dimensions of each output
- Which invisibles do we make visible?
 - Testability = visibility + control
 - Which internal states does our technology allow us to examine?

Comparison data:

IF detection is via comparison to oracle data, where do we get the data?

- Historical results
 - actual behavior of a previous test
- Values computed by a human judge and stored for reference
- Values computed from a reference program
- Values computed from a model
- Values computed from a simulation
- Value acceptable-ranges computed from model / etc.

Detection

What heuristics/rules tell us there **might** be a problem?

- Comparator (processor of comparison data)
- Comparator (processor of a model, such as evaluating a rule or heuristic that output X should not exceed output Y)
- Diagnostic message (including exception or log output of a probe) caught by automaton
- Unacceptable behavior
- Usually-unacceptable behavior that needs further evaluation
- Unexpected or improbable behavior

Evaluation:

How do we **decide** whether X is a problem or not?

- Are known errors dealt with by modifying a failing test so that it no longer interferes with the test run, or does the test generate the equivalent of an exception that can be checked against a list of known failures (and that serves as a central point for update when bugs are allegedly fixed)
- Under what circumstances is a behavior determined to be a definite fail?
 - Does execution halt to support troubleshooting?
 - Does execution repeat to demonstrate reproducibility?
- Is the default assumption that suspect behavior is probably a fail?
 - What of the maybe-it-failed tests? How are they processed?

Troubleshooting support

Failure triggers what further data collection?

- What state information (system / program / data) is known at the start of the test? Is it available on failure?
- What state information can be collected on failure?
- To what extent does collection of information item X change what we will learn when we look at item Y (Heisenberg applied to testing)
- What diagnostics are triggered by failure
- Automated replication?
- Automated replication with parametric variation? (is this something that can be requested for difficult bug?)
- Is longer-sequence history relevant / available?

Notification

How/when is failure reported?

- halt and wait for a human to check on status
- sound alarm
- automatic email to the owner of the test run or of the code (or other interested parties)?
- automatic bug report?
- attachments include dump of ... ???

Retention

In general, what data do we keep?

What data?

- status information
 - what tests / what results / when run
 - what results per test across what version
 - how many bugs this iteration
- failure logs and supporting data
- configuration logs for each test run (enable evaluation of why an existing bug didn't show up in previous test)

Why do we keep it?

- for external review
 - process auditors
 - litigation support
 - buyer of product
 - buyer of development company
- for status reporting
- for troubleshooting support
- for technical support (failure / success data)
- for marketing support (e.g. examples of demonstrator scenarios that work)

Maintenance

How are tests / suites updated / replaced?

- Mechanism for updating tests
 - rewrite the entire test?
 - rewrite methods that map to program behavior that changed
 - revise stored data
- How often is maintenance needed
 - how fragile are the underlying models
 - how fragile is the test code / documentation
- Define "inertia" as opposite to agile "velocity". What is the inertia caused by the test suite / tool?

Relevant contexts:

Under what circumstances is this approach relevant/desirable?

- Suitable staff for development / maintenance of tests?
- Timeframe for development (e.g. 6 weeks before release is unsuitable for test tool that needs 3 months of work)
- Timeframe for recovery of automation-development and automation-maintenance costs
- Appropriate assumptions of stability of code / design / platform, etc.
- Extent to which the testing requirements are managed by adding this testing

Closing Thoughts

- Many people in our field are trapped in commodity roles:
 - The style of testing often promoted as "professional" is adversarial, inefficient, relatively unskilled, and easy to outsource
 - The style of test automation most often promoted is automated execution of regression tests, which are narrow in scope, redundant with prior work
- Especially in difficult economic times, it is important for:
 - testers to ask how they differentiate their own skills, knowledge, attitudes and techniques from commodity-level testers
 - test clients to ask how they can maximize the value of the testing they are paying for, by improving their focus on the problems most important to the enterprise
- In this talk, we look at testing as an analytic activity that helps the other stakeholders understand the subject domain (here, investing), the models they are building in it, and the utility of those models and the code that expresses them. We see lots of test automation, but no regression testing.
- Rather than letting yourself get stuck in an overstaffed, underpaid, low-skill area of our field, it makes more sense to ask how, in **your application's** particular domain, you can use tools to maximize value and minimize risk.