



Automating Production Simulations for Added Value

Created for:

TestNet

Spring Event 2013

Nieuwegein, Netherlands

By:

Scott Barber

Chief Technologist

PerfTestPlus, Inc.

Nothing can stop automation



STONE AGE



BRONZE AGE



IRON AGE



DARK AGE



MODERN AGE



COMPUTER AGE

Logistics & Introductions

Before we get started:

Logistics

- schedule / snacks
- session management / participation

Who am I?

- some general background
- industry background
- what do I know about test automation?!?

Who are you?

- role
- test automation experience

THE PESSIMIST

THE GLASS IS HALF EMPTY.



THE OPTIMIST

THE GLASS IS HALF FULL.



MIDDLE MANAGEMENT

THE GLASS IS WHATEVER THE BOSS SAYS IT IS.



THE BUSINESS ANALYST

LET'S FIND OUT WHAT THE INDUSTRY SAYS THE GLASS IS.



THE QA TESTER

THE GLASS SPILLS WHEN I TRY TO USE IT.



THE PROGRAMMER

I WISH THAT WAS A BEER.



Current Automation Value

To help me tune this class, I'd like to know:

What do you Automate?

What value do you get from that Automation?

What *doesn't* your Automation do that you'd like it to?

Automation Evolution

Historically “common” Frameworks:

1. **Linear Scripting** (record/playback)
2. **Script Libraries** (+ edit/modularize)
3. **Data-Driven** (+ variable data)
4. **Keyword Driven** (“recording” abstracted)
5. **Hybrid** (you guessed it)
6. **Modularity** (script libraries revisited supporting hybrid)
7. **Business Process** (user flow focus enabled by modularity)

Where is your org?

Automation “Frameworks”

Linear Scripting:

Tester manually records each step & inserts Checkpoints. Then “plays back” the recording.

Advantages

- Automation expertise not required
- Fast & simple (when it works)

Disadvantages

- Disposable scripts
- Test data difficult to parameterize

Automation “Frameworks”

Script Libraries:

Scripts initially recorded by “Record & Playback” method, then common tasks inside scripts are identified and grouped into Functions. Functions then called by main test script in different sequences.

Advantages

- Higher level of code reuse
- Easier Script Maintenance

Disadvantages

- Some technical expertise is necessary
- More time is needed to create scripts.
- Test data difficult to parameterize

Automation “Frameworks”

Data-Driven:

Navigation logic in Test Scripts, Test Data abstracted to external files and to populate script variables. Variables are used both for Input and Verification. Scripting can be either Linear or Library.

Advantages

- Changes to Scripts do not affect Data
- A variety of scenarios can be executed by changing the external file

Disadvantages

- More time is needed to create Scripts and Data

Automation “Frameworks”

Keyword Driven:

Requires development of data tables and keywords, independent of the test automation tool used to execute them . Tests can be designed with or without the Application. Functionality of the app is documented in a table with step-by-step instructions.

Advantages

- Provides high code re-usability
- **Tool agnostic**
- Tests can be designed in advance

Disadvantages

- Initial investment is high
- **High automation expertise is required**

Automation “Frameworks”

Hybrid:

A combination of one or more of the previous frameworks. Most frameworks get here whether deliberate or not.

Advantages

- Highly custom
- Don't get here unless someone loves maintaining it

Disadvantages

- Highly custom
- “Someone” stops loving to maintain it

Automation “Frameworks”

Modular:

Seriously, this is just script libraries in OO Language vs. Structured Programming Language – but now with data-driven, keywords, and/or action words.

Advantages

- Should be today’s starting point (but it isn’t)
- Done well, enables GUI, API, & Unit-level automation under one “framework”

Disadvantages

- Few testers are good at the coding part
- Few developers are good at the test design part

Automation “Frameworks”

Business Process:

Breaks up large Business Processes into Components for re-use. Really, it's just Modular, except the abstraction is based on common user actions instead of common segments of code.

Advantages

- Some people like business language over code jargon
- **Actually useful for working with SMEs & end-users**

Disadvantages

- Some people like code language over business jargon
- **Makes devs want to run away again.**

Automation “Frameworks”

What's the Point?

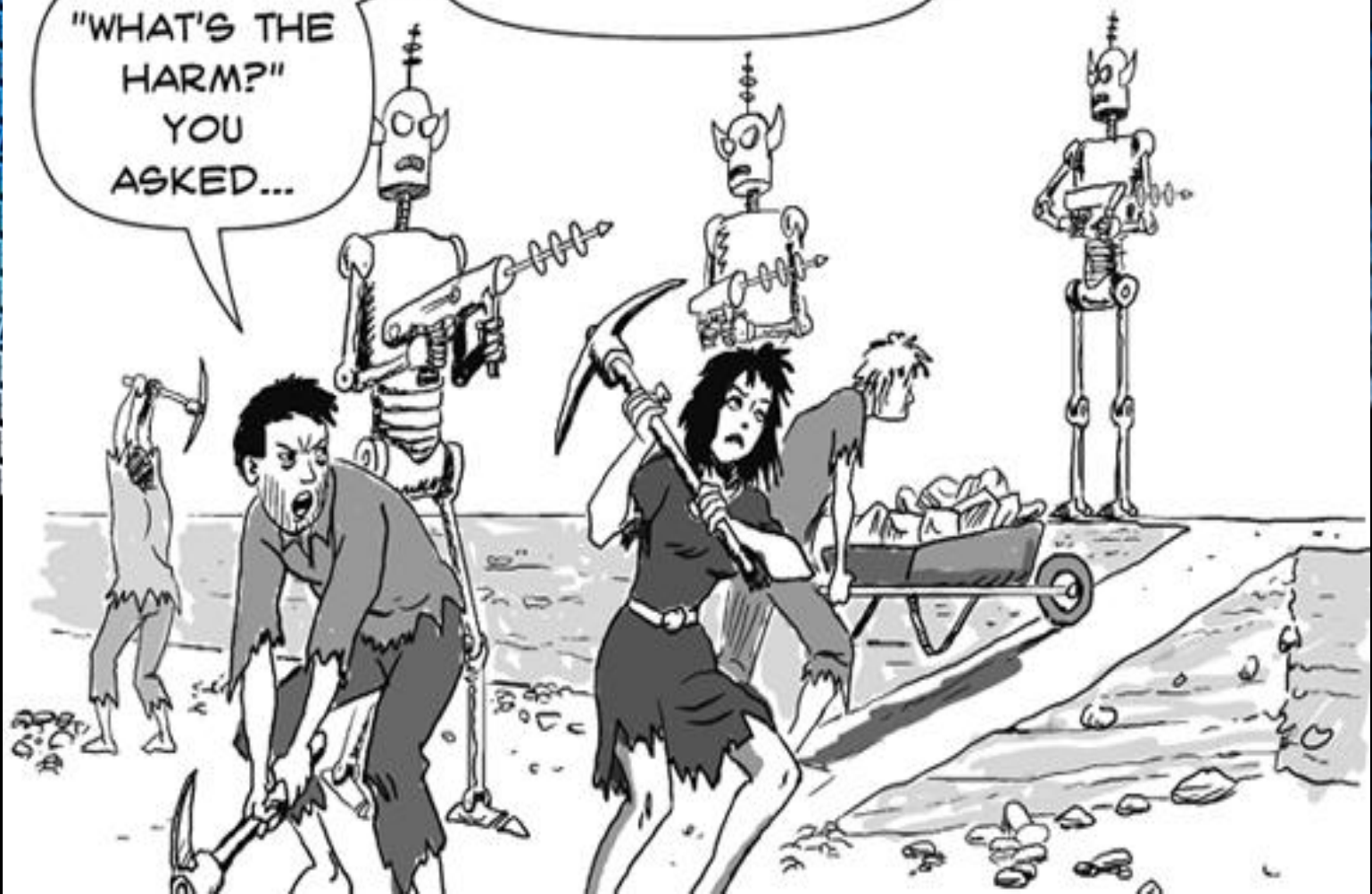
**I don't care
what “framework” you
use, they all
miss something
important!**

FIRST WE
AUTOMATED THE
TESTING...

THEN WE AUTOMATED
THE DEVELOPMENT...

THEN WE AUTOMATED
THE DESIGN.

"WHAT'S THE
HARM?"
YOU
ASKED...



Is it fair to say...

Most test automation is designed from:

Code (unit tests)

Stories (acceptance tests)

Requirements/Specs (checking)

Use/Business Cases (expected flows)

Previous Defects (regression)

Perceived Risk (risk-based)

Experience (tribal knowledge / thin air)

Is it also fair to say...

Most test automation exercises:

I/O of:

- Units
- Functions/Procedures/Objects
- 3rd Party Components

Calculations & Algorithms:

“Pre-Commit” Data Validation:

A limited selection of Usage Paths:

And most test automation requires:

Narrowly defined Oracles to detect...

Almost anything of value!

And can I go so far as...

Most test automation ignores:

The Pesticide Paradox:

- Same test, data & sequence => no new info
- “The code” learns to pass tests
- “Doers do what checkers check”

Human Variability:

- People rarely do what we expect...
- And almost never read the manual

System Variability:

- Testing on a “clean” system is bogus
- The system will be anything but “clean” in Prod

Brian Marick agrees

“An automated test's value is mostly unrelated to the specific purpose for which it was written. **It's the accidental things that count**: the untargeted bugs that it finds.”

And goes on to say...

“Automated tests, if written well, can be run in sequence, and **the ordering can vary from day to day**. This can be an inexpensive way to create something like **task-driven tests** from a set of feature tests... automated tests can **take advantage of randomness** (both in ordering and generating inputs) **better than humans can**.”

Excerpts from: When Should a Test Be Automated?

Brian Marick, 1998

Where Am I Going With This?

Don't stop
just because your
automated checks work.
Add more value with
Production
Simulations

Production Simulations, huh?!?

Production Simulations strive to imitate Production Conditions, such as:

- Real user-like navigation
- Realistic data variance
- Time variability between transactions
- Human reaction to errors/error messages
- Multi-Device usage
- Abandonment / Connection disruption
- Malicious behaviour
- Client & Server “noise”

Pause for Prerequisites

Object Orientation of Scripts:

- Separate scripts will **not** work for dynamic navigation
(unless you've automated the scripting, but even still, just don't)
- OO concepts are key, not OO dogma
(It doesn't matter if your "objects" are modules, methods, business procedures, functions or subroutines – just abstract logically & usefully)
- OO will minimize impact of code changes
(not negate it, just make it manageable)
- It's ok to "mix & match" libraries, modules, functions & business objects
(I don't care **what** your university professor would say)

Pause for Prerequisites

External Variable Data I/O:

- Ideally, you want to be able to modify your data “on-the-fly”
(Or at least change seed values for randomization routines)
- You will need to be able to write to your data file “on-the-fly”
(Trust me, if you are truly randomizing, random data, you’ll need to know what got accepted later in the script)
- Resetting the database after each execution negates much of the value
(So get your own data instance)
- Keywords, OS commands, delay times, etc. all count as “external variable data”
(Yes, you may use multiple data files)

Pause for Prerequisites

Be able to detect and handle application errors:

- If execution halts at each error, you won't get very far
(At least, not if you're doing it right)
- Many “on error handlers” in your automation will have random, conditional logic
(Sometimes try again, sometimes try again with different data, sometimes hit the back button, etc.)

Be able to drive the app from multiple levels:

- You will want to be able to switch between GUI & API & direct DB connect during execution
(I know, crazy, right?!?)

**Before I get ahead of myself...
Who has questions?**



Designing Production Simulations

Production Simulation Design “steps”:

1. **B**uild/Obtain Model(s)
2. **I**dentify Path(s)
3. **D**etermine Variance
4. **E**stablish Randomization
5. **T**rending Considerations
6. **I**nter Coding Logic
7. **M**ake Sure it Works
8. **E**xecute

Build/Obtain Models

Models of Interest

System Usage

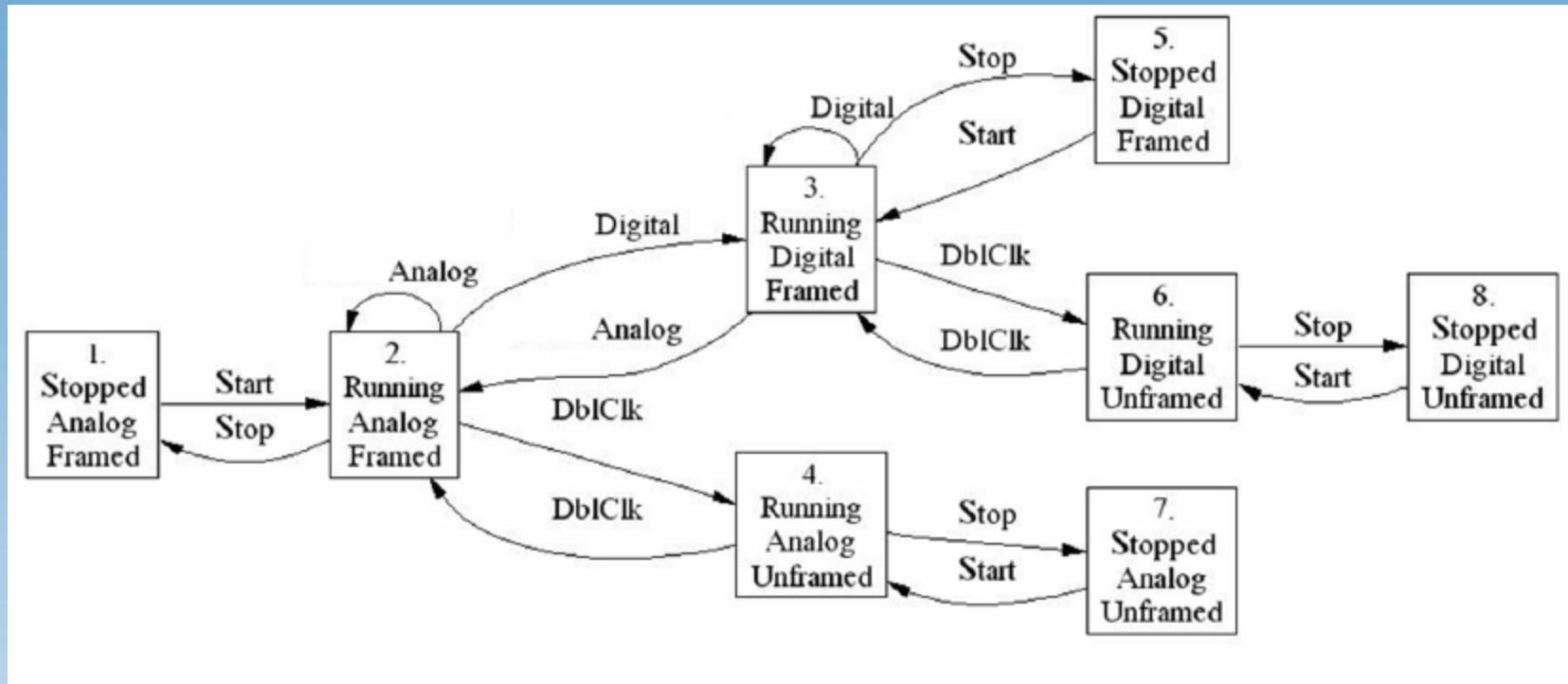
System Architecture

Network Architecture

Database/Datastore Structures

Build/Obtain Models

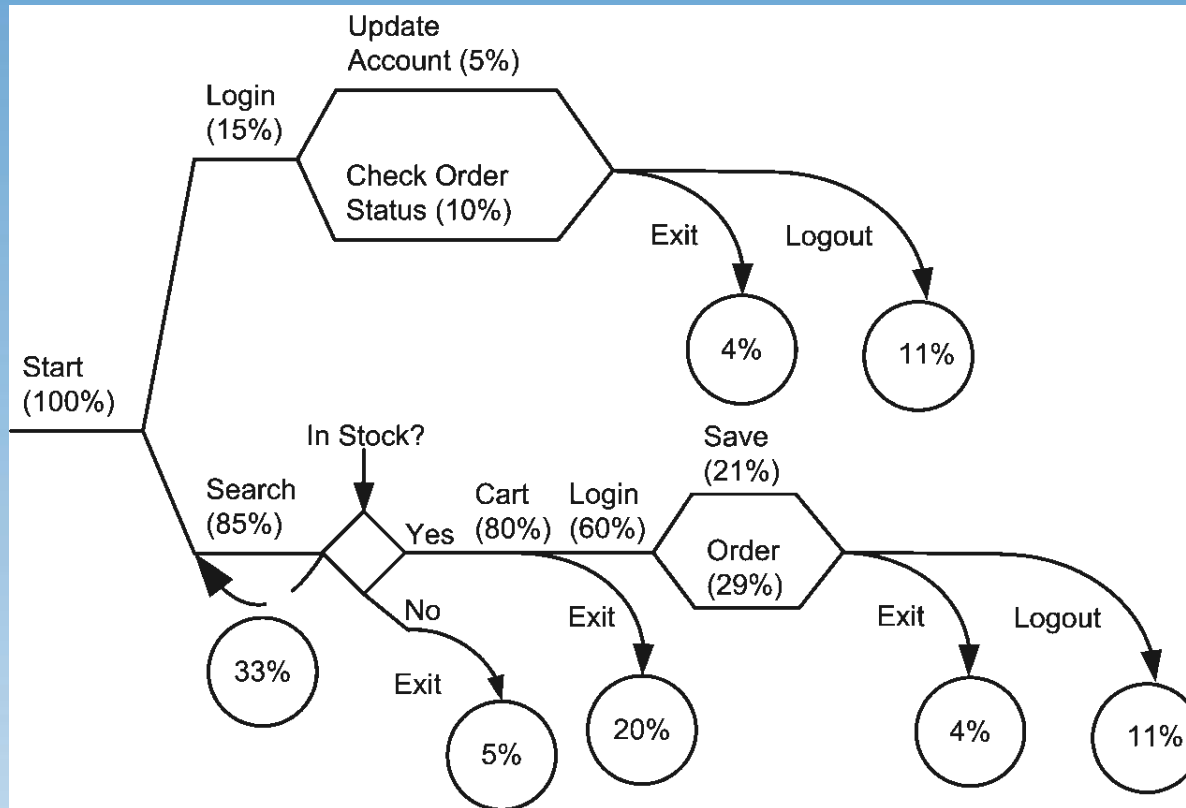
System Usage Model Example



State Transition Diagram

Build/Obtain Models

System Usage Model Example



UCML Diagram

Build/Obtain Models

Other System Usage Modelling Methods

UML

Decision Tables

Combinations

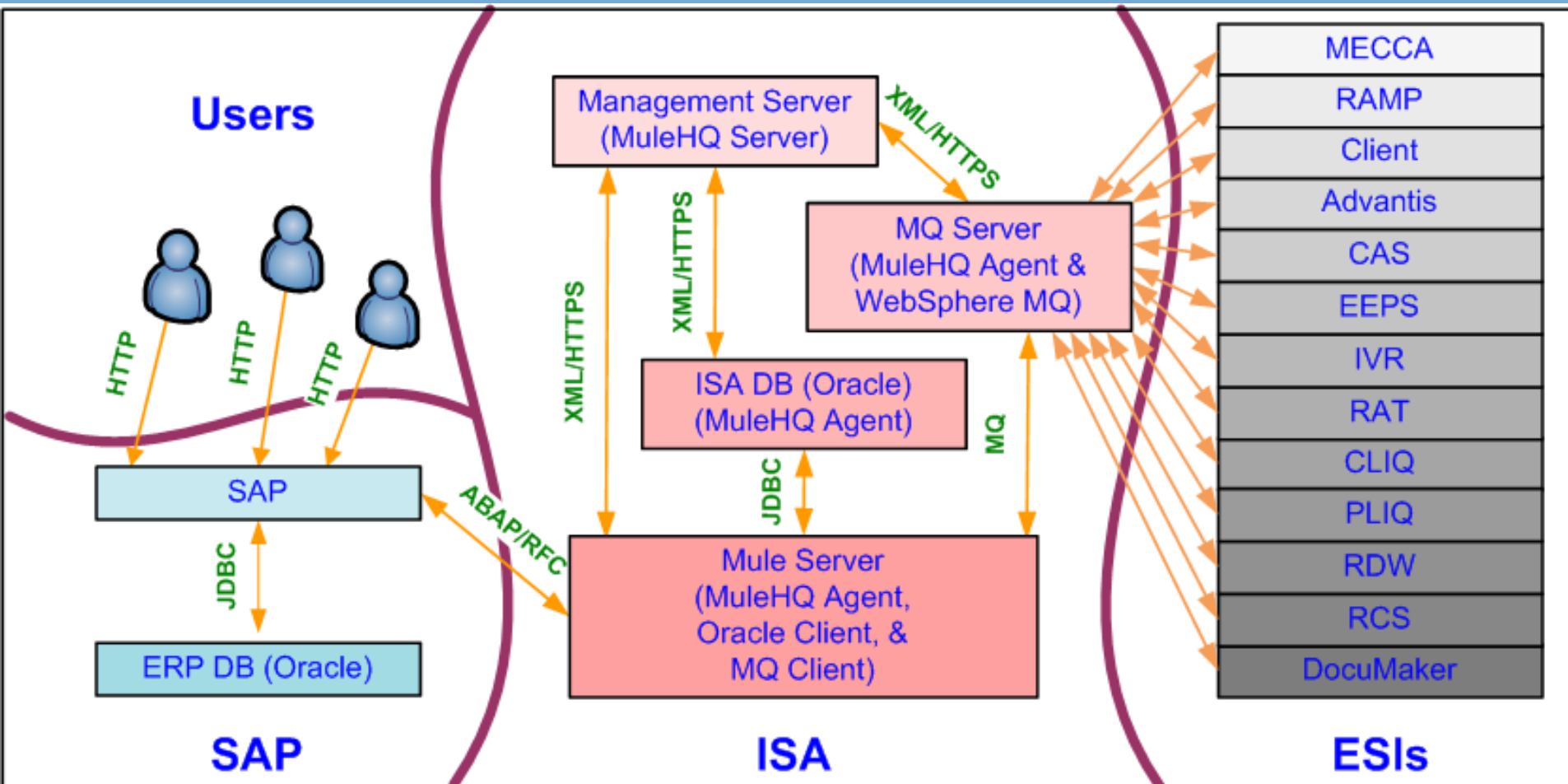
Markov Chains

Grammars

Combinations

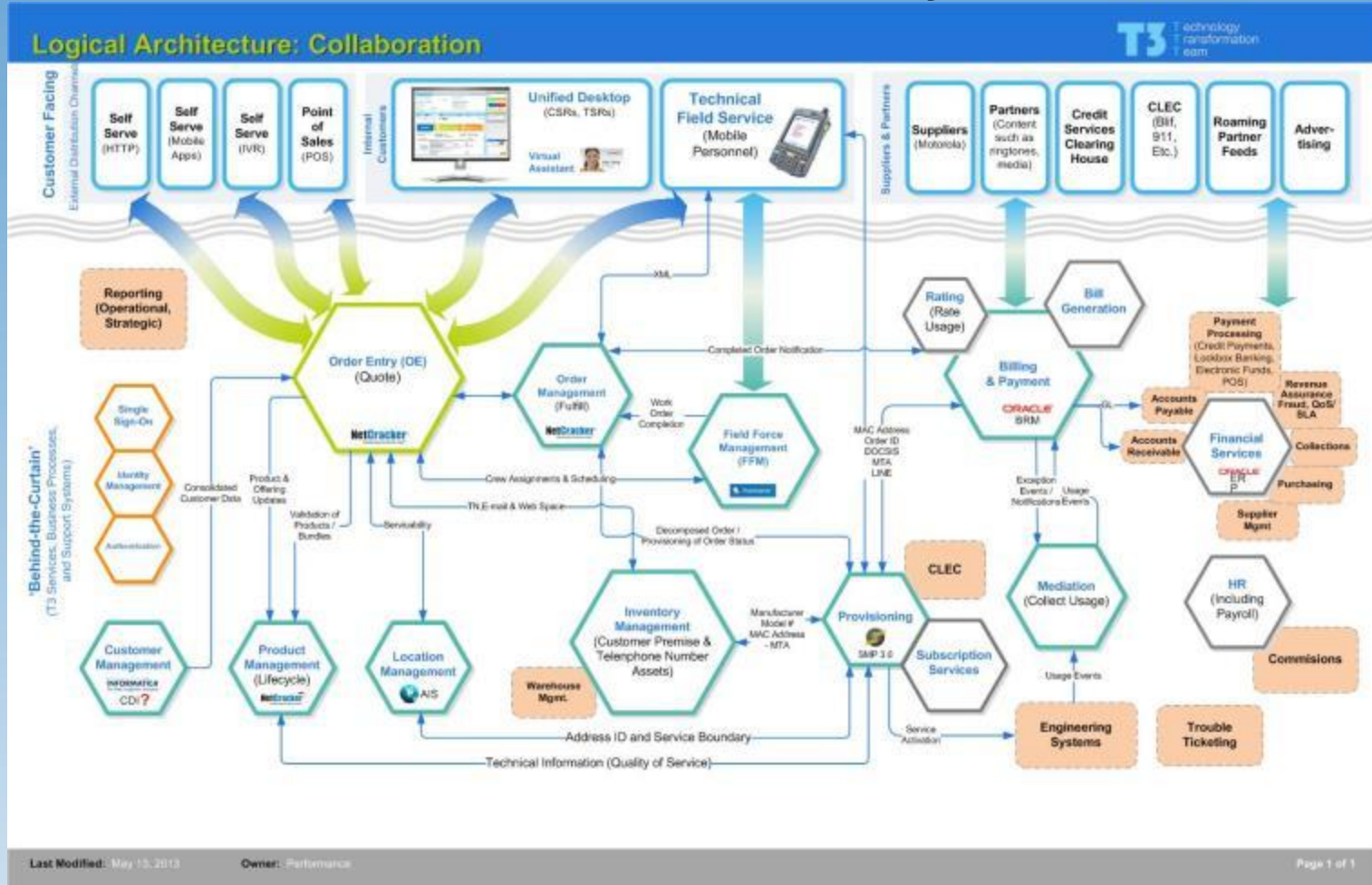
Build/Obtain Models

System Architecture Model



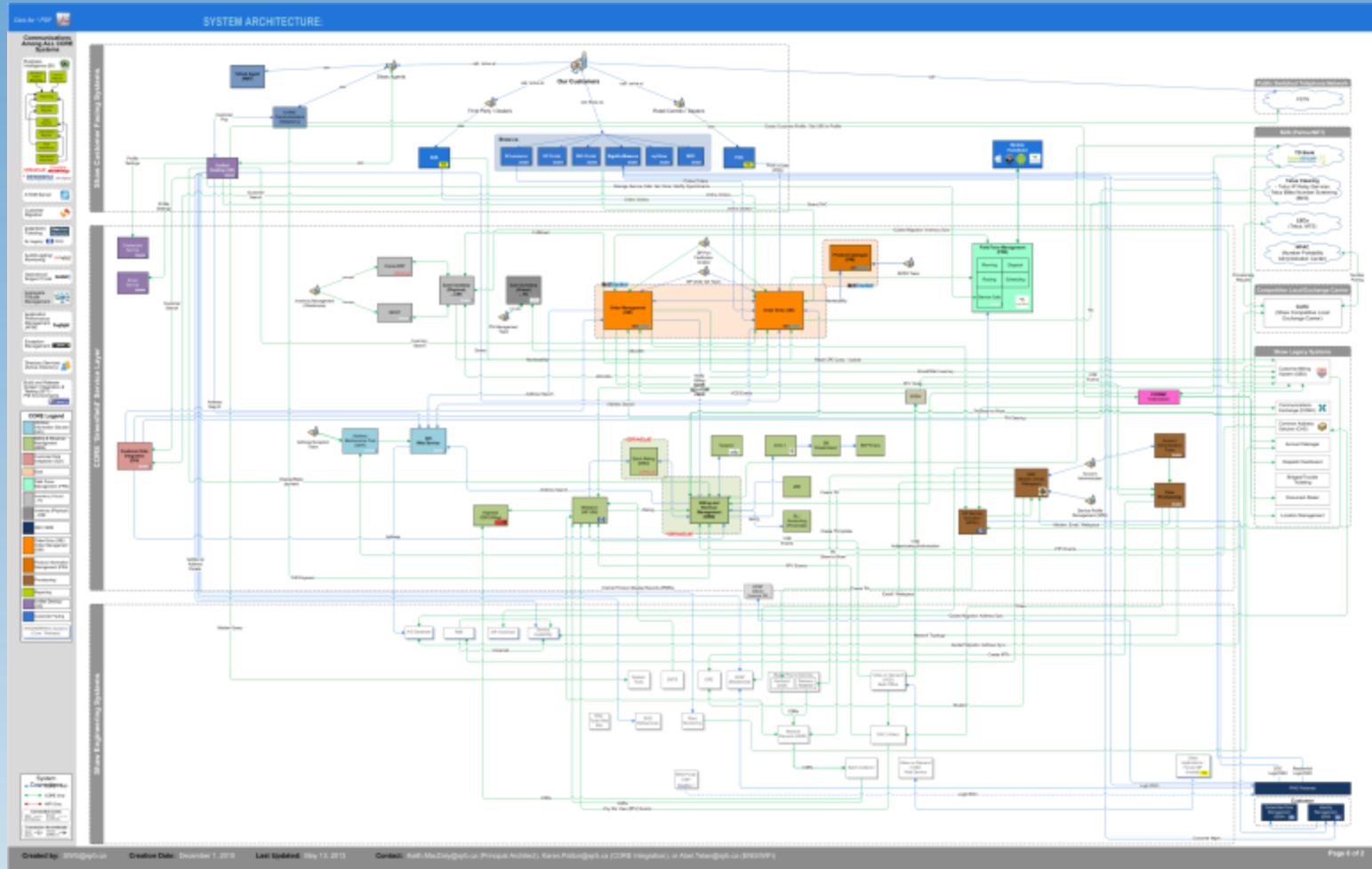
Build/Obtain Models

System/Network Architecture Hybrid



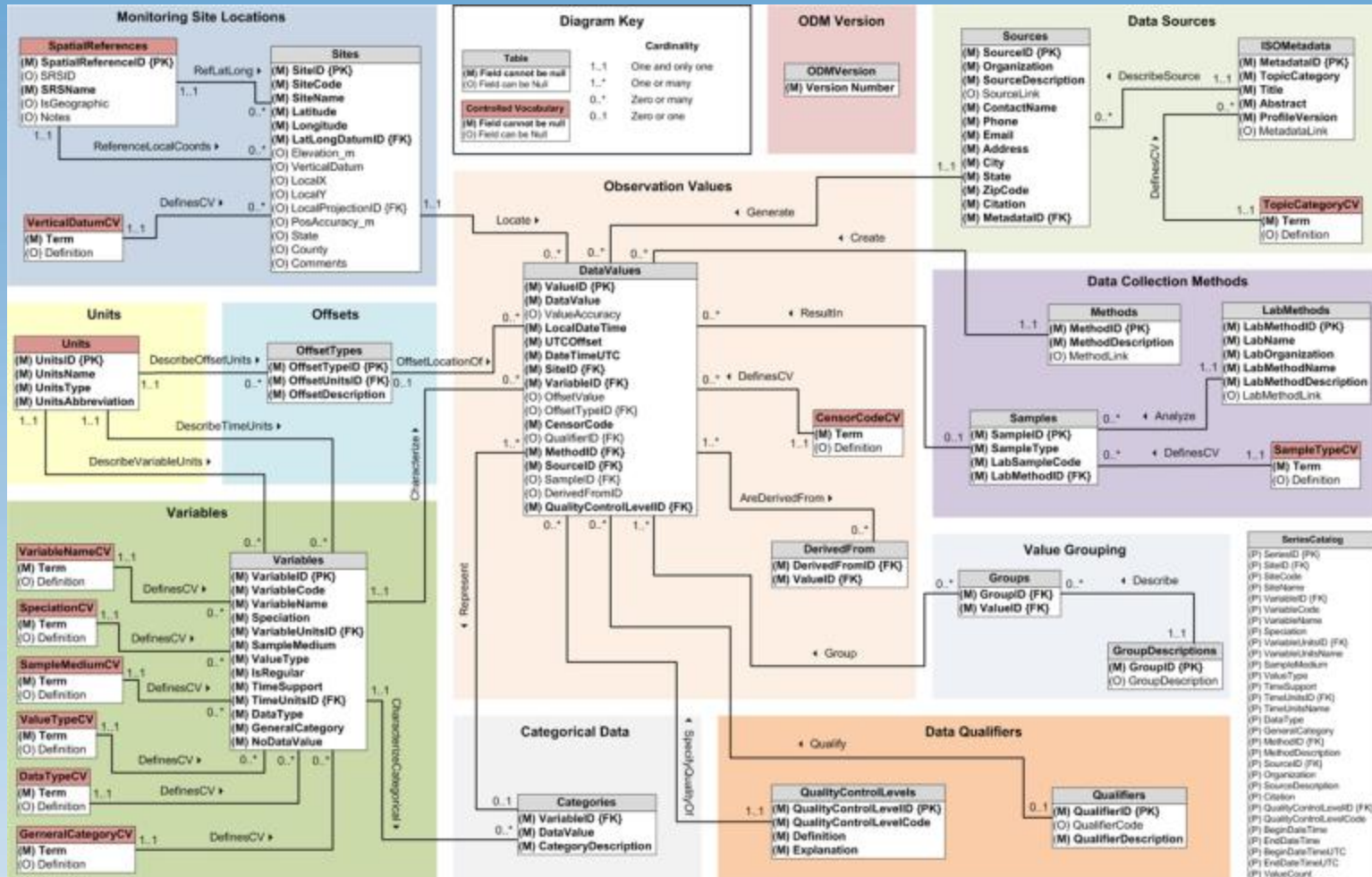
Build/Obtain Models

Network Architecture Model



Build/Obtain Models

Data Model



Identify Paths

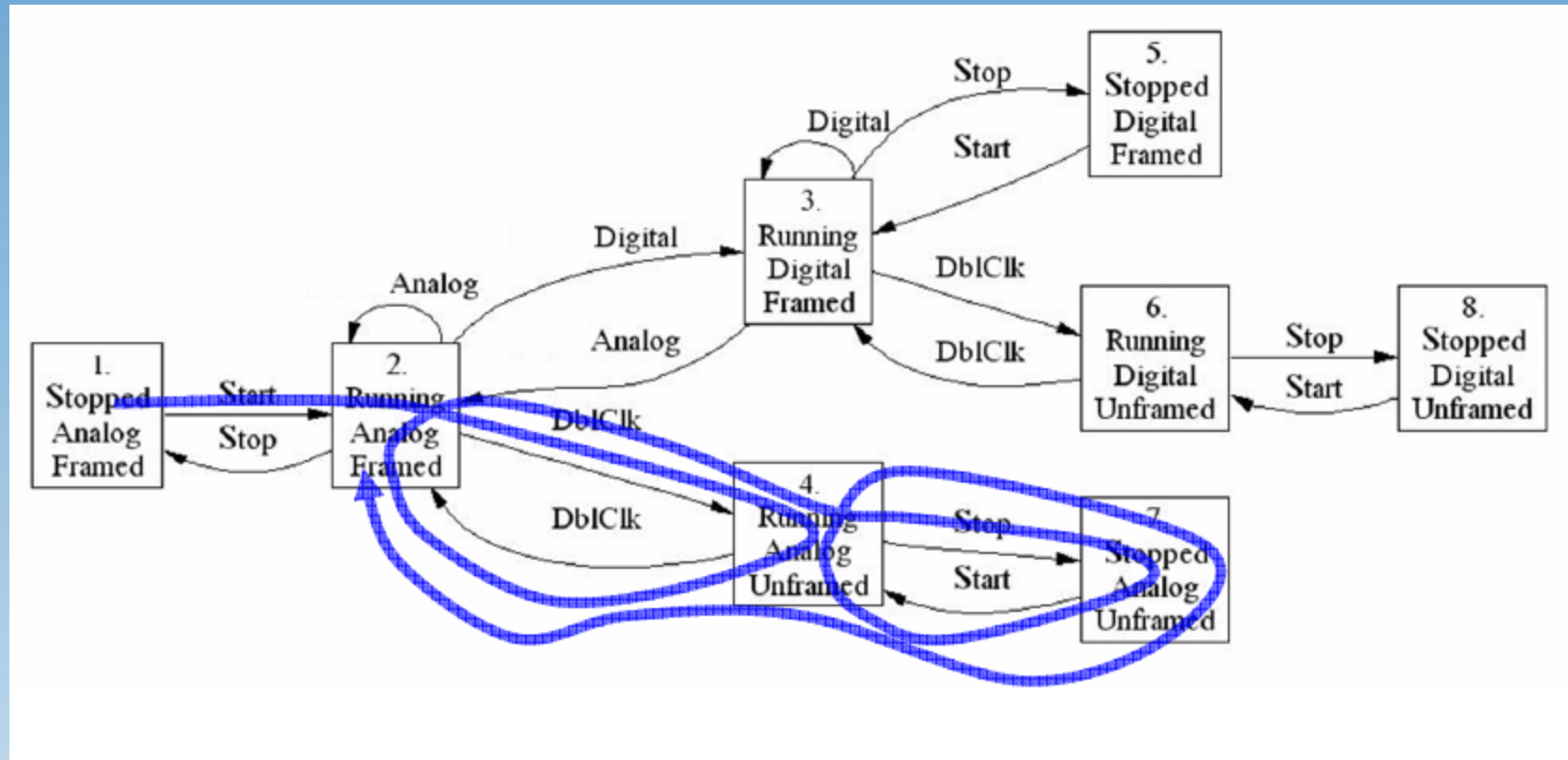
Paths of Interest

Navigation

Data

Identify Paths

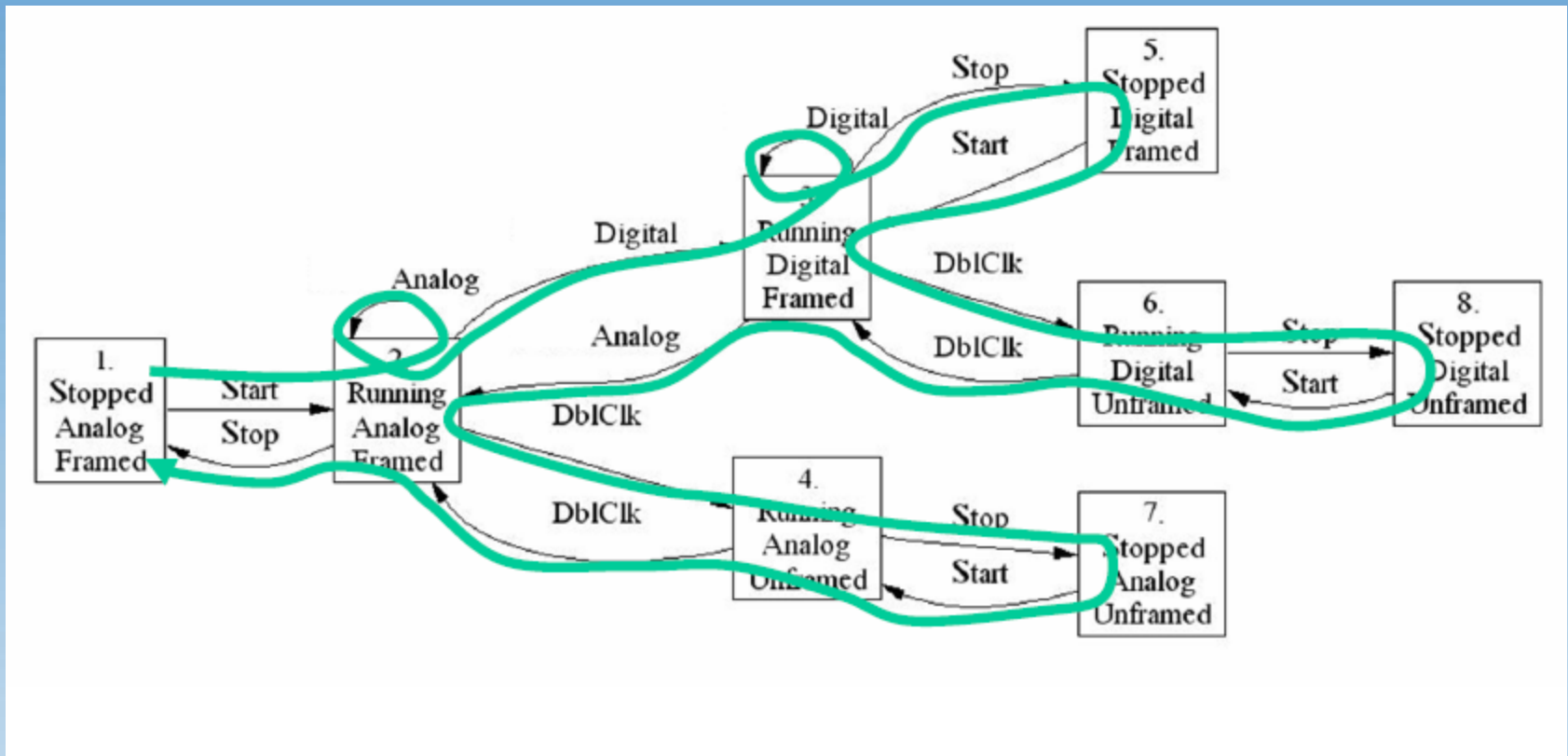
Navigation



A Random Walk

Identify Paths

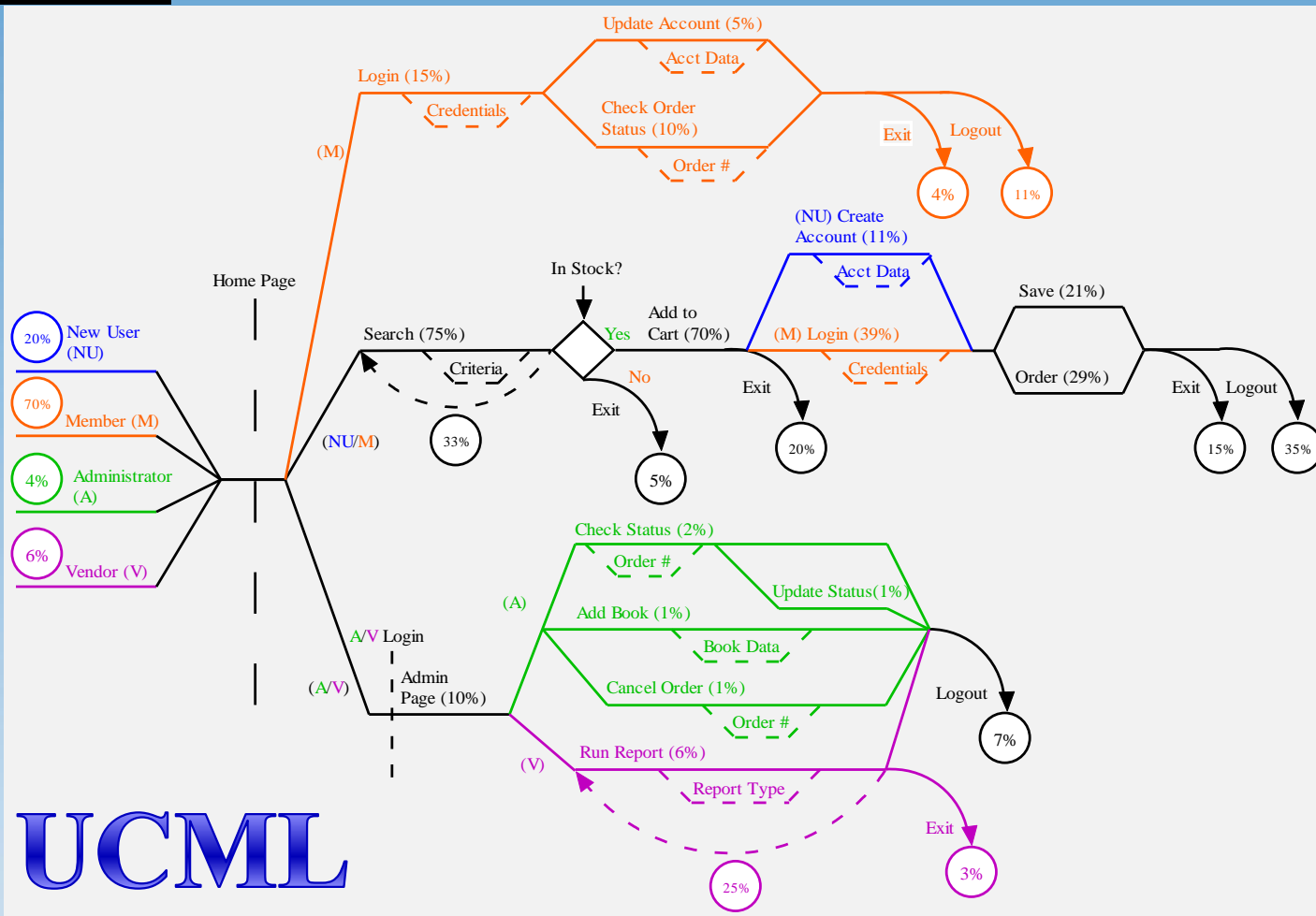
Navigation



All Transitions

Identify Paths

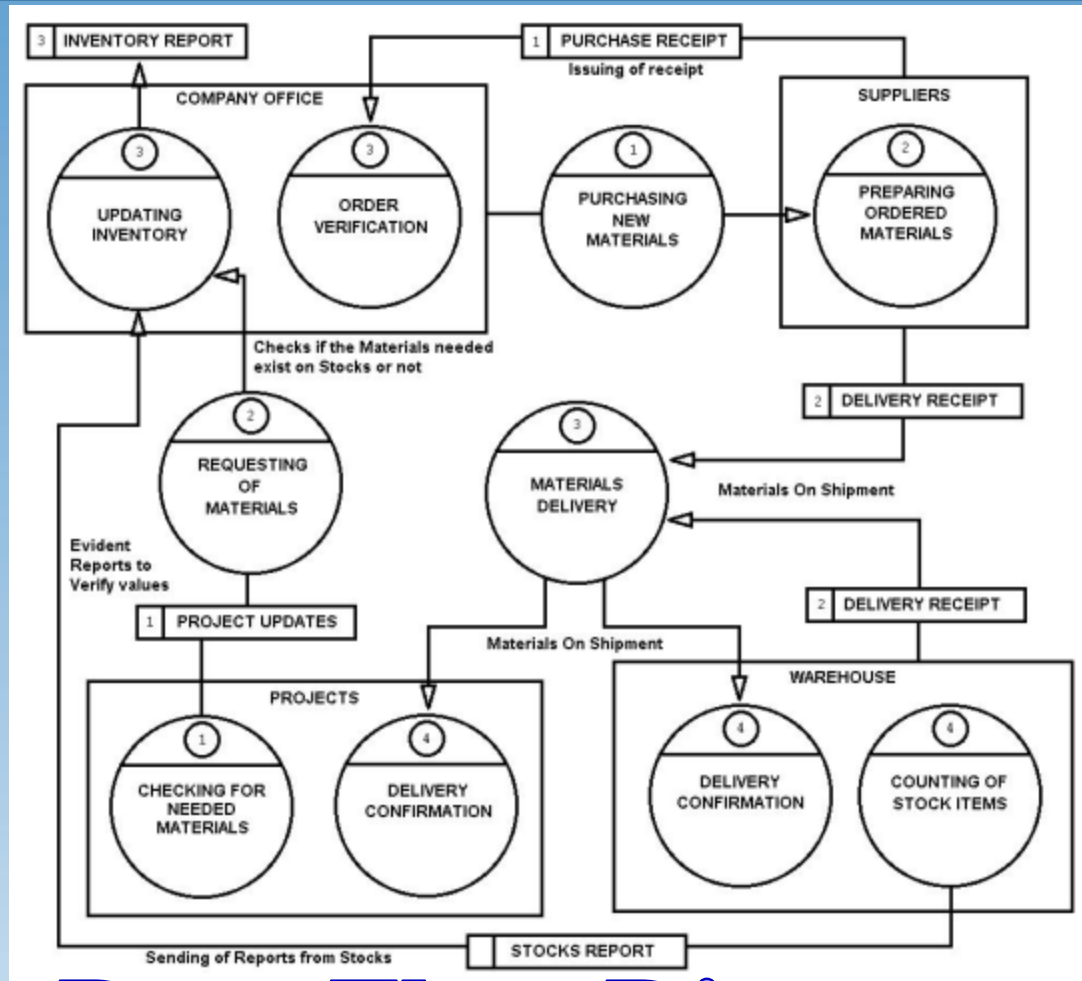
Navigation



UCMIL

Identify Paths

Data



Data Flow Diagram

Determine Variance

Variance of Interest

Data

Delays

Client “Noise”

Server “Noise”

Determine Variance

Delays & Abandonment

- Static delays yield unrealistic results
(a range of +/- 50% is typically adequate)
- Delays between each page should be different
(users do not spend the same amount of time on every page)
- Don't let every path run to completion
(not every user will finish what they started)
- Determine how long they think
 - Log files
 - Industry research
 - Observation
 - Educated guess/Intuition
 - Combinations are best

Determine Variance

Delays

Every page needs a think time – after you determine the think time for that page, document it.

These think times should cause your script to pace like real users.

Event Type	Event Name	Type	Min	Max	Std	Req't	Goal
Procedure name: Initial Navigation()							
Timer name:	tmr_home_page	negexp	4	N/A	N/A	8	5
Timer name:	tmr_login	normdist	2	18	4.5	8	5
Timer name:	tmr_page1	linear	5	35	N/A	8	5
Timer name:	tmr_data_entry	negexp	8	N/A	N/A	8	5
Timer name:	tmr_page2	normdist	3	9	3	5	3
Timer name:	tmr_submit_transaction	linear	2	4	N/A	5	3
Timer name:	tmr_signout	N/A	N/A	N/A	N/A	8	5

Determine Variance

Abandonment

If a page takes too long to display, users will eventually abandon your site – typically leaving hanging sessions, dead objects, etc.

Not simulating abandonment makes your test appear to indicate acceptable behavior.

Page Name	Abandonment Distribution	Abandonment Min Time	Absolute Abandonment
Home Page	Normal	5 sec	30 sec
Pay Bill	Uniform	10 sec	240 sec
Search Web	Negexp	8 sec	30 sec
Submit Taxes	Inverse Negexp	30 sec	900 sec
Validate Field	Normal	5.5 sec	20 sec

Determine Variance

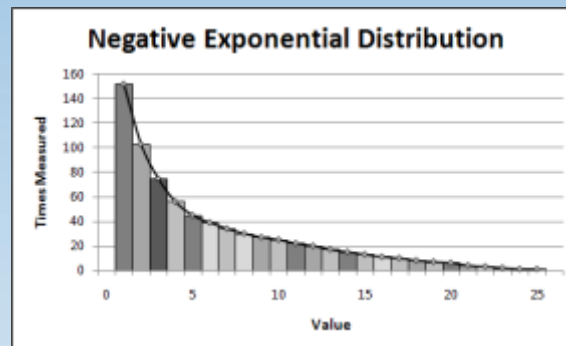
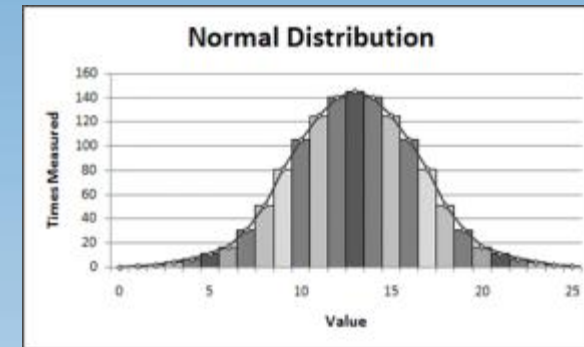
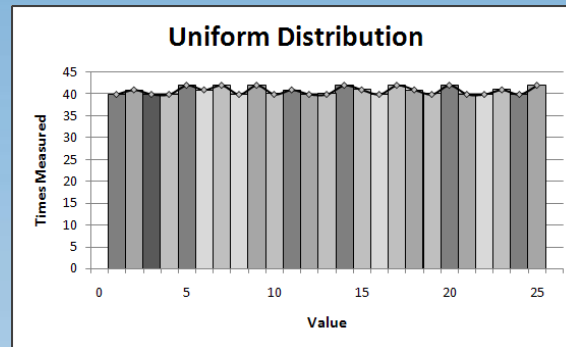
Noise

- Users won't run your app in isolation
 - you shouldn't either
 - launch Word & extra browser tabs & email & IM periodically
 - and close them
- Servers won't be clean either
 - Anti-virus programs & auto-updates & dynamic resource re-allocation (oh my!)
 - Turn logging on... then off... then on again
- Internet / Wireless connections choke & drop
 - so make them choke & drop & recover

Establish Randomization

Randomization Considerations

- **Boundaries**
 - High & low will do
 - Don't use the same highs & lows for everything (rhythmic stuff is unrealistic)
- **Patterns**
 - Uniform
 - Normal
 - NegExp ("long tail")
 - Double Hump Normal



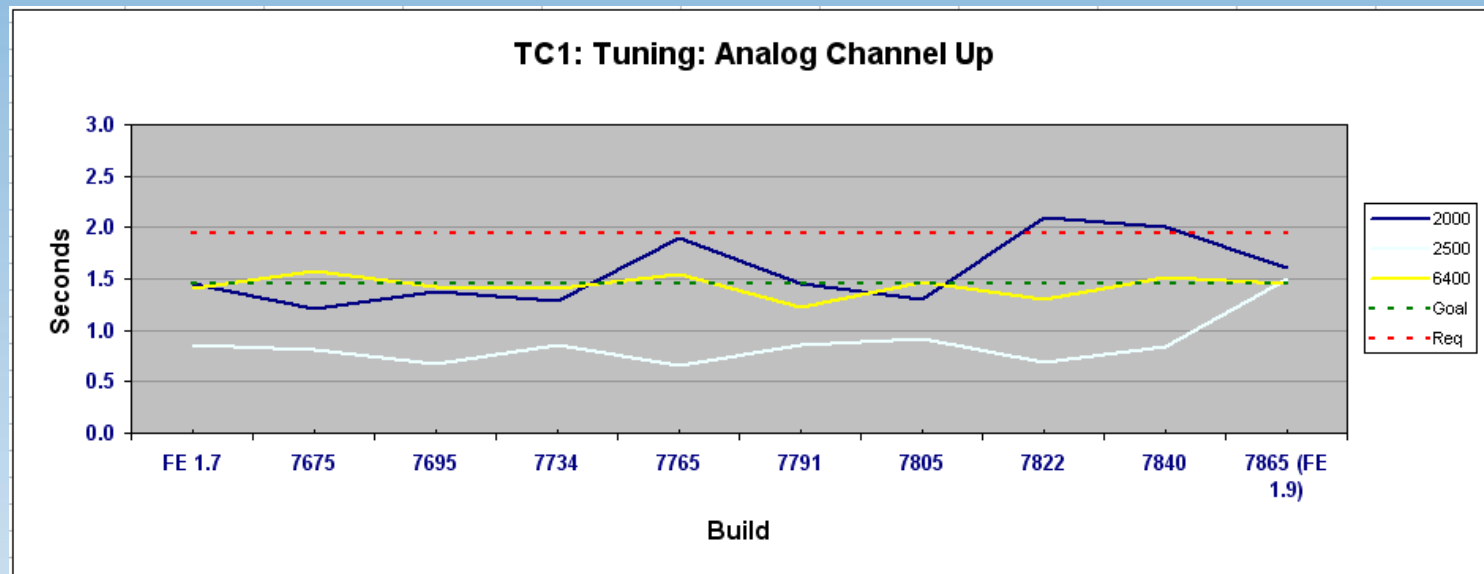
Trending Considerations

Trend Data to Collect

Response Times

Resource Utilization

Noise Impact



Insert Coding Logic

'nuff said

Make Sure it Works

**Don't forget to
test your test!**

Execute

**Grab a beer &
watch it go!**

In Summary

Automation Frameworks keep “evolving”

- But only in terms of variable/object abstraction

Most Test Automation is “low-volume checking”

- Not bad, but sad, since computers can do *so* much more!

Production Simulations add additional value

- Many bugs get detected only under prod-like conditions
- Injecting just a little realism can go a long way!

Modelling & Thinking like a Performance Tester help

- Thinking about modelling probably helps more

At least some of this should be easy to implement, so...

- **Try it & tell me how it goes!!!**

Questions?





Chief Technologist, PerfTestPlus, Inc.

sbarber@perftestplus.com

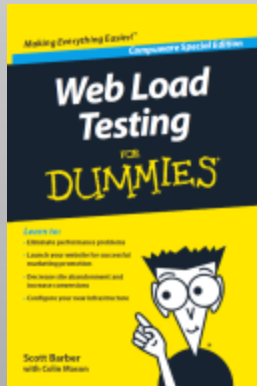
www.perftestplus.com

[@sbarber](https://twitter.com/sbarber)

Co-Founder: Workshop On Performance and Reliability

www.performance-workshop.org

Author:



Co-Author:



Contributing Author:



Books: www.perftestplus.com/pubs

About me: about.me/scott.barber

Contact Info

Scott Barber

about.me/scott.barber

Chief Technologist
PerfTestPlus, Inc.

E-mail:

sbarber@perftestplus.com

Web Site:

www.PerfTestPlus.com

Blog:

scott-barber.blogspot.com

Twitter:

[@sbarber](https://twitter.com/sbarber)

Appendix: UCML 1.1™ Syntax

Agenda

- Introduction and Learning Objectives
- UCML™ Overview
- Symbols
- Combining Symbols
- Interactive Demonstration
- Supplementary Information
- Summary
- Where to go for more Information
- Questions/Contact Info

Introduction and Learning Objectives

Introduction

- Development History
- Use History

Learning Objectives

- The value of representing workload distributions visually.
- What UCML™ is and is not designed to be used for.
- To create UCML™ diagrams.
- To use these diagrams to aid in data gathering.
- To supplement UCML™ diagrams with a simple spreadsheet containing all the information required to create accurate performance test scripts.

UCML 1.1™ Overview

UCML™ is a set of symbols that can be used to create visual system usage models and depict associated parameters.

Intended Use:

- Visually depict system with multiple usage paths and/or users.
- Document system usage and associated parameters.
- UCML™ is a modeling framework, not a standard.

Advantages of Use:


- Easier to create than many other methods.
- Intuitively understood by all members of the team.
- Gives all team members a common language to discuss usage models.

UCML 1.1™ Overview

Applicability to Performance Testing

- Supplement or replace workload distributions, operational profiles, pivot tables, matrixes, and Markov chains.
- Make it possible to use the same model when discussing the system with both technical and non-technical team members.
- Develop automated test scripts from the model with no intermediate steps.
- Eliminate need for separate documentation of Performance Test Scenarios

UCML™ Symbols

 UCML™ Symbols represent possible paths through the system during a specific period of time.

Quantity Circle

- “How Many?” or “How Often?”
- Finite numbers or percentages enclosed by a circle.



UCML™ Symbols

Description Line

- Solid horizontal lines that represent activities and user types.
- Label indicates user type, activity, or navigation path.
- Percentages indicate the frequency the activity or user type.



Search Titles (15%)

UCML™ Symbols

Synchronization Point

- Used to depict convergence in...
 - Time
 - Navigational Location
- Dashed vertical line, typically with name above.

Sync Point



UCML™ Symbols

Option Box

- Show optional activities or varied data along a common path.
- Represented by a dashed box suspended from a description line.
- Labeled to describes the option or data variance.
- Metadata described in associated spreadsheet.

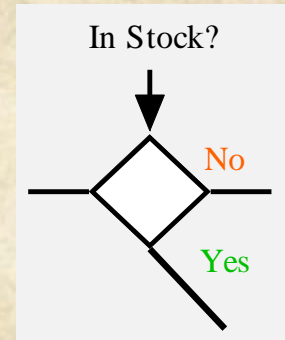
Order Book (15%)

◆ ◆ Pick Title ◆ ◆
◆ ◆

UCML™ Symbols

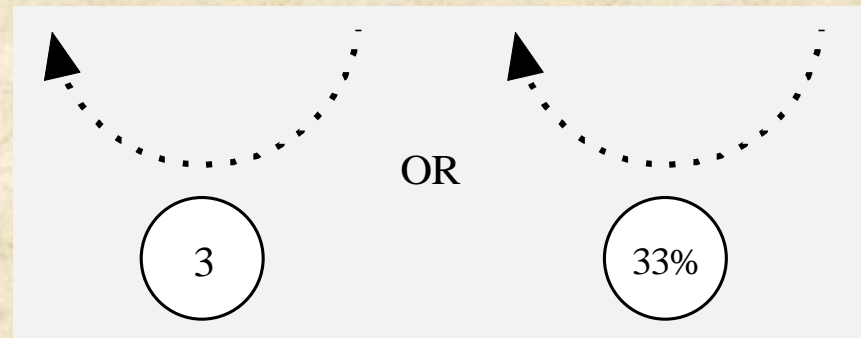
Condition

- Represent navigational decision points based on returned data.



Loop

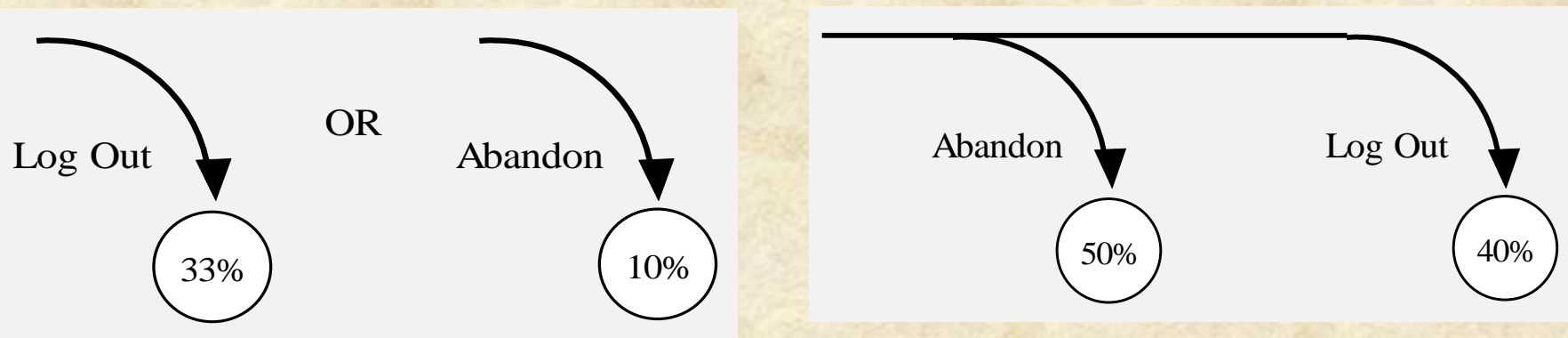
- A dashed semicircle represents a loop on the navigation path, where the activity encircled by the semicircle may be repeated.
- Number of iterations or the percentage of users repeating the activity is indicated in a quantity circle.



UCML™ Symbols

Exit Path

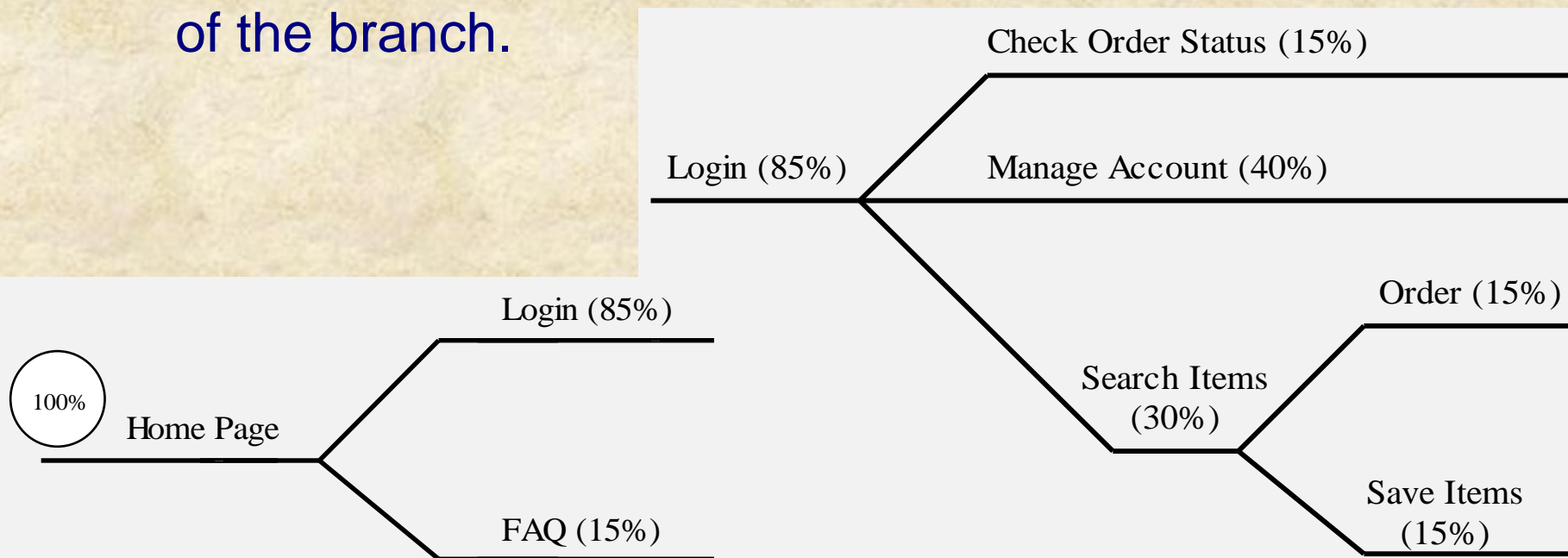
- A downward-curving line pointing to a quantity circle indicates the number or percentage of users leaving the system at a specific point on the path.
- All entering users must exit.
- The type of exit should be noted by a label.



UCML™ Symbols

Branch

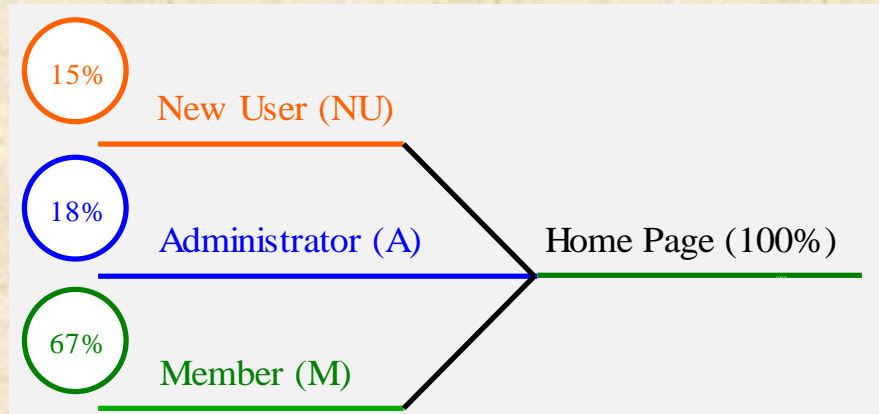
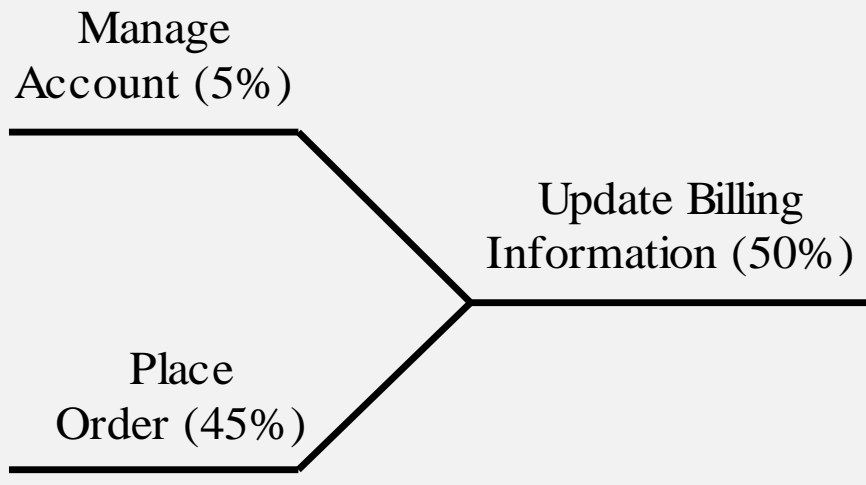
- Represents an intersection where the user can choose among multiple paths.
- Quantity of users must be the same on both sides of the branch.



UCML™ Symbols

Merge

- Inverse of a branch.
- May color code by user type.



UCML™ Symbols

Summary

- Symbol libraries for MS Visio and SmartDraw available at <http://www.perftestplus.com>.
- Symbols are intended to be enabling and flexible, not limiting.
- Symbols are a framework, not a standard.
- If existing framework limits you, add your own symbols (and please email me so I can consider including your symbols in the next version – with proper accreditation).

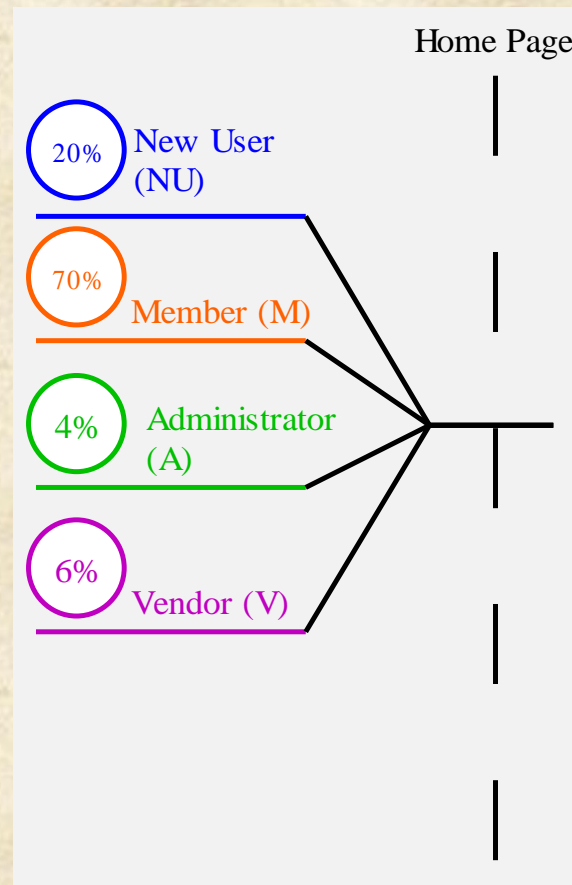
Combining Symbols into a Model

Overview

- Combining symbols is what makes UCML™ powerful.
- Symbols are combined to represent actual usage, not an approximation due to modeling limitations.
- Don't feel limited by published examples, it's good to combine symbols creatively (as long as it intuitive represents reality).
- Don't hesitate to estimate or guess when you don't know – it's easy to distribute the model to gather additional information.
- Remember that all users who enter the system must be accounted for across every vertical slice of the model.

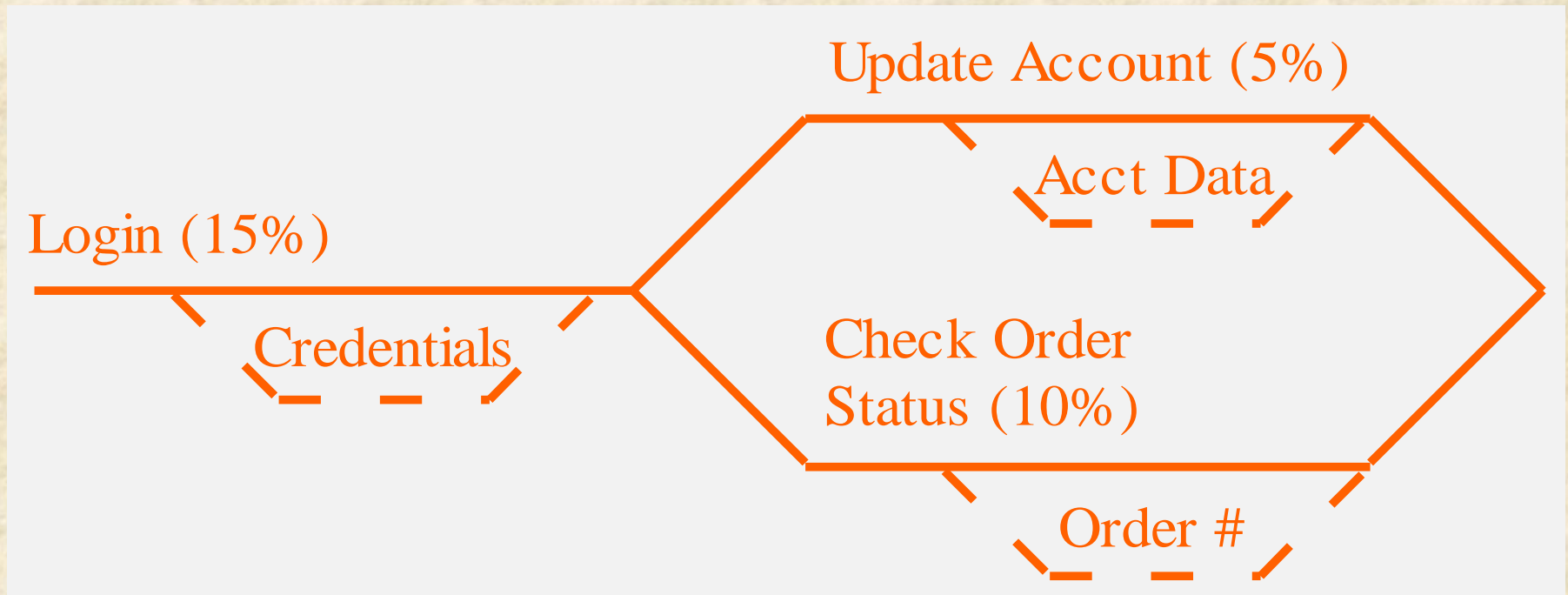
Combining Symbols into a Model

Example – System Entry



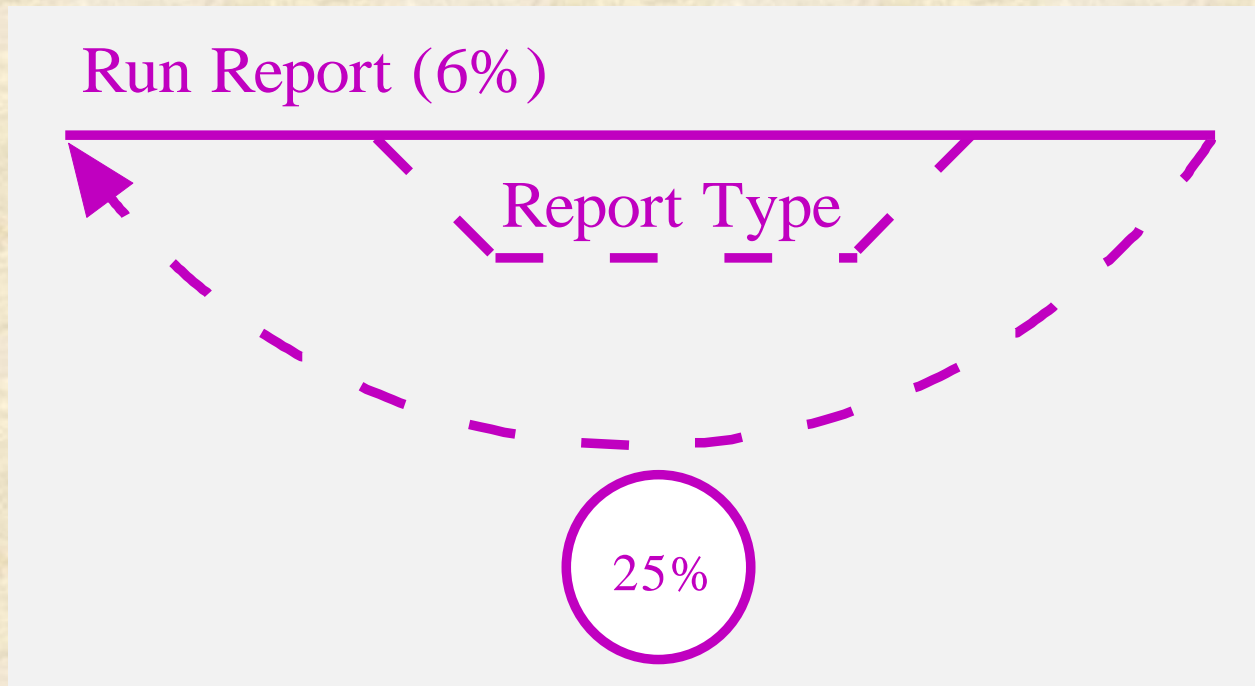
Combining Symbols into a Model

Example – Navigate with Options



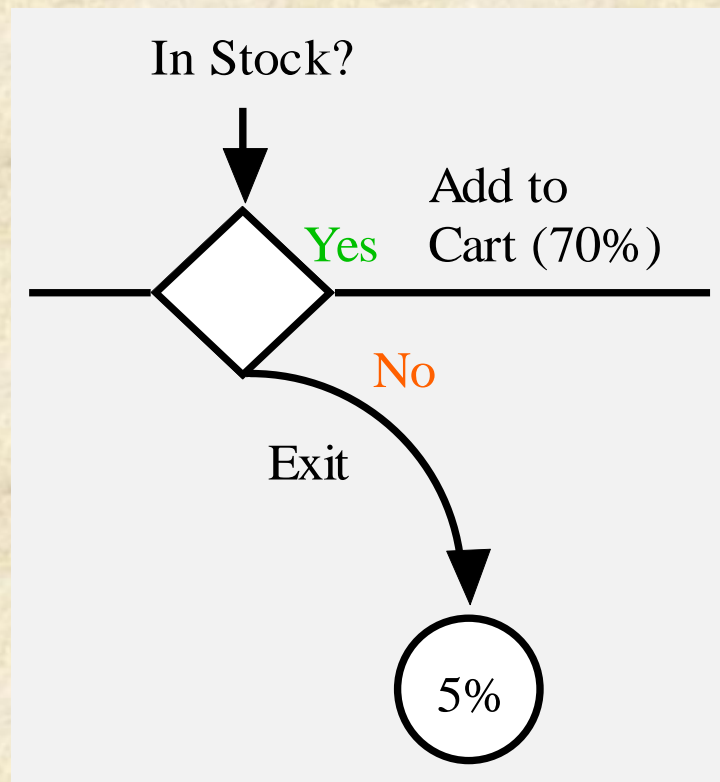
Combining Symbols into a Model

Example - Looping



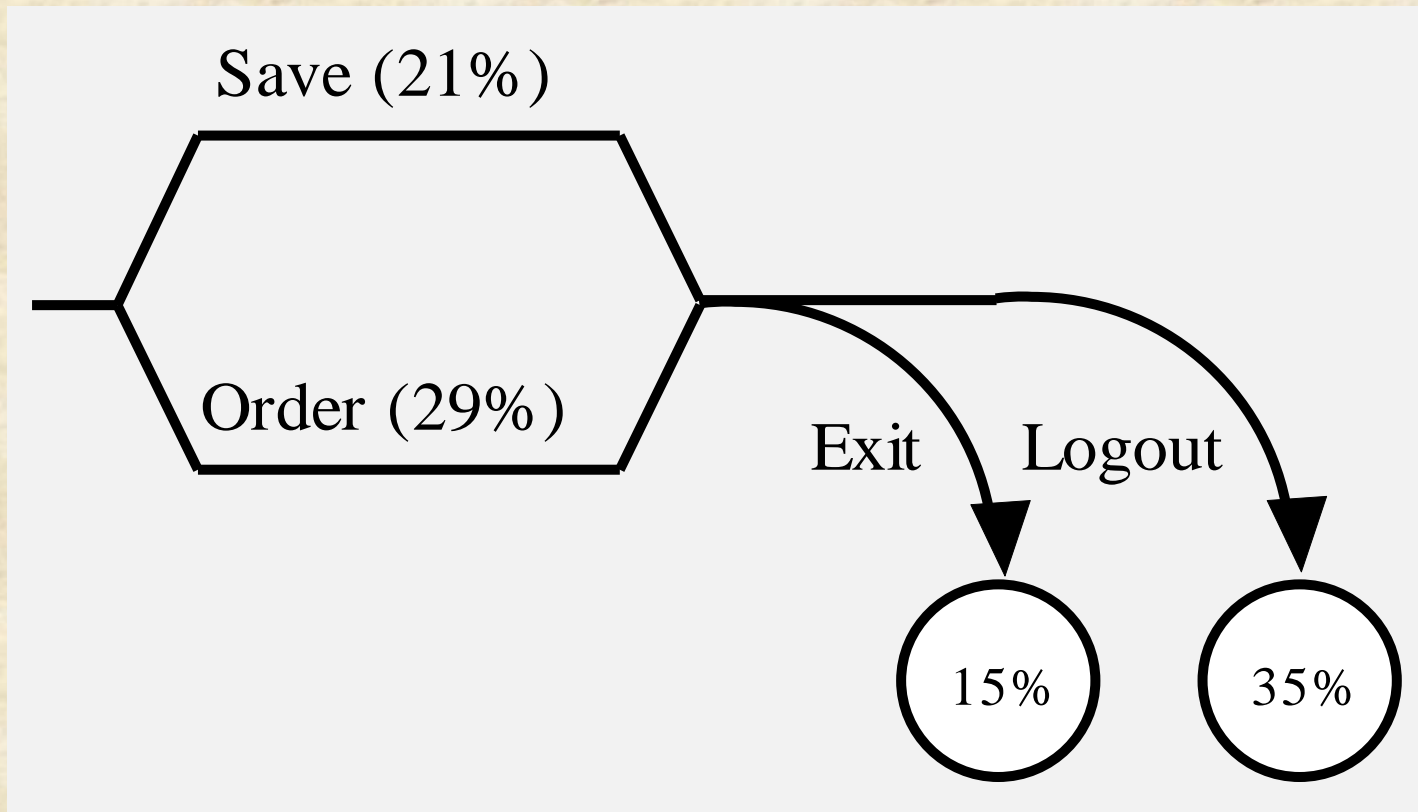
Combining Symbols into a Model

Example – Conditional Navigation



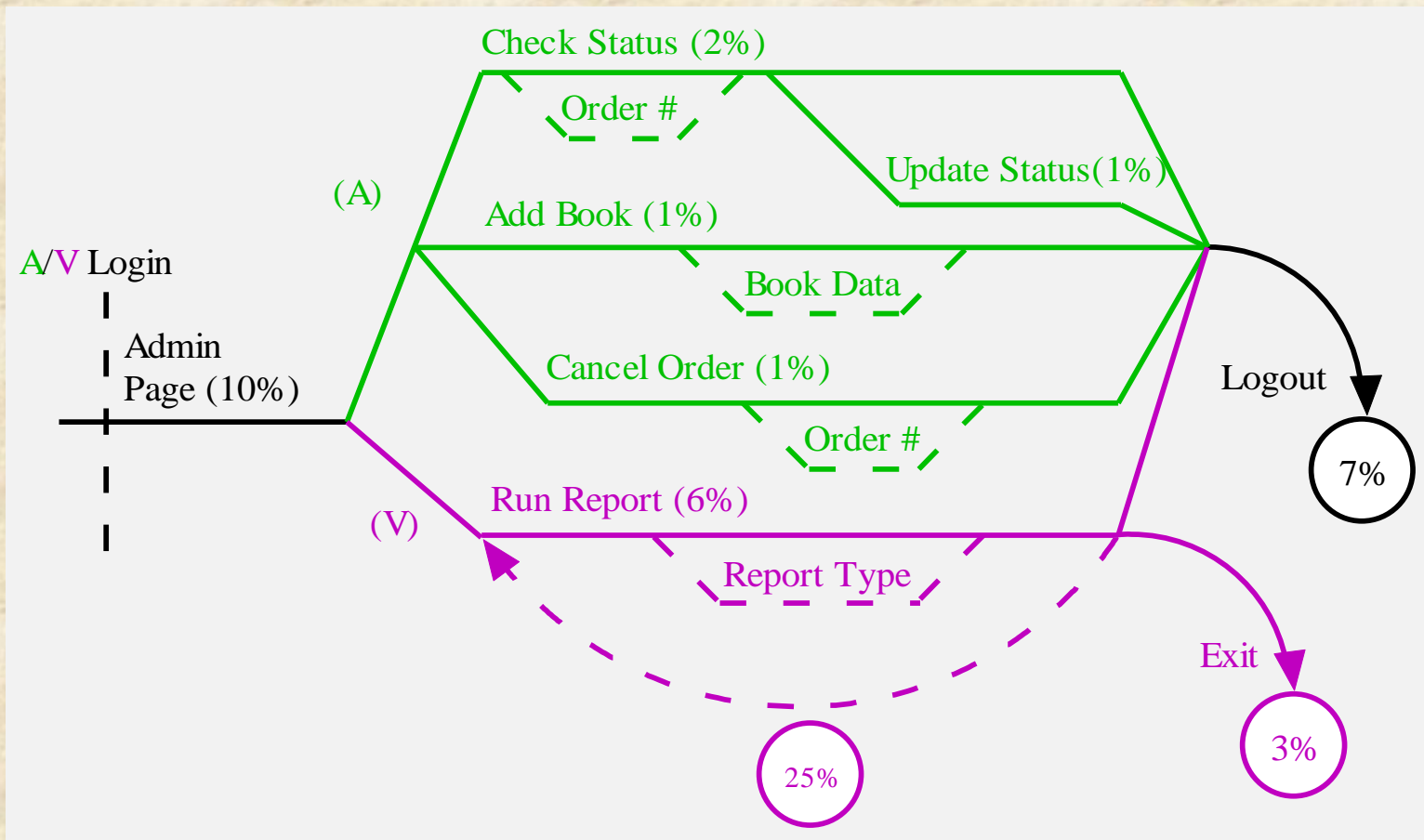
Combining Symbols into a Model

Example – Exit System



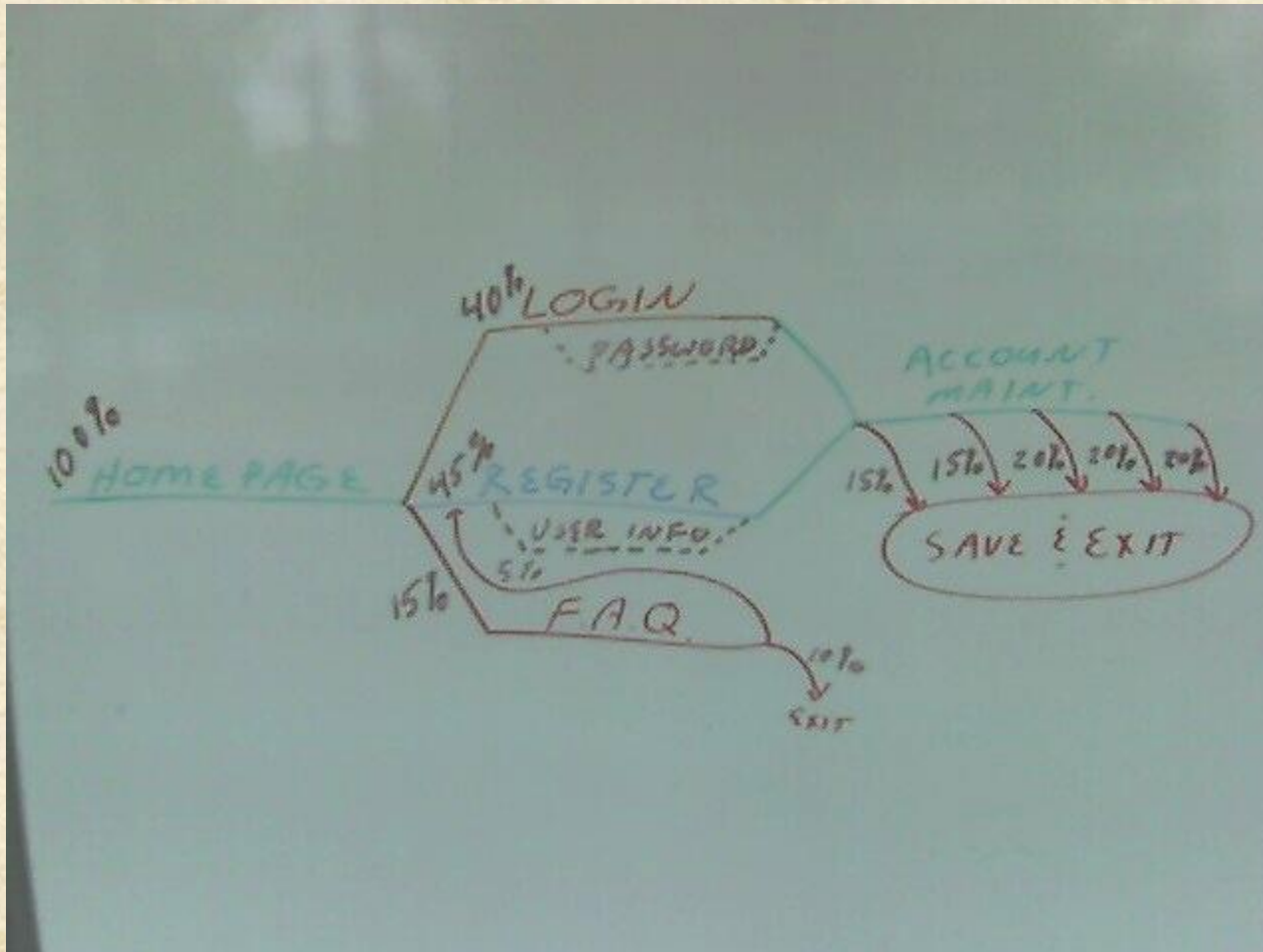
Combining Symbols into a Model

Example – Putting it all together



Combining Symbols into a Model

The first UCML model



Interactive Exercise (On-Line Book Store)

Users:

- Four Types: New(20%), Members(70%), Admins(4%), Vendors(6%)
- All Enter through HomePage

New Users and Members:

- Search by Title, Author or Keyword
- Add books to cart, save cart

New Users – Create Account (i.e. become member)

Members – Login, Order, Update Account, Check Order Status

Interactive Exercise (On-Line Book Store)

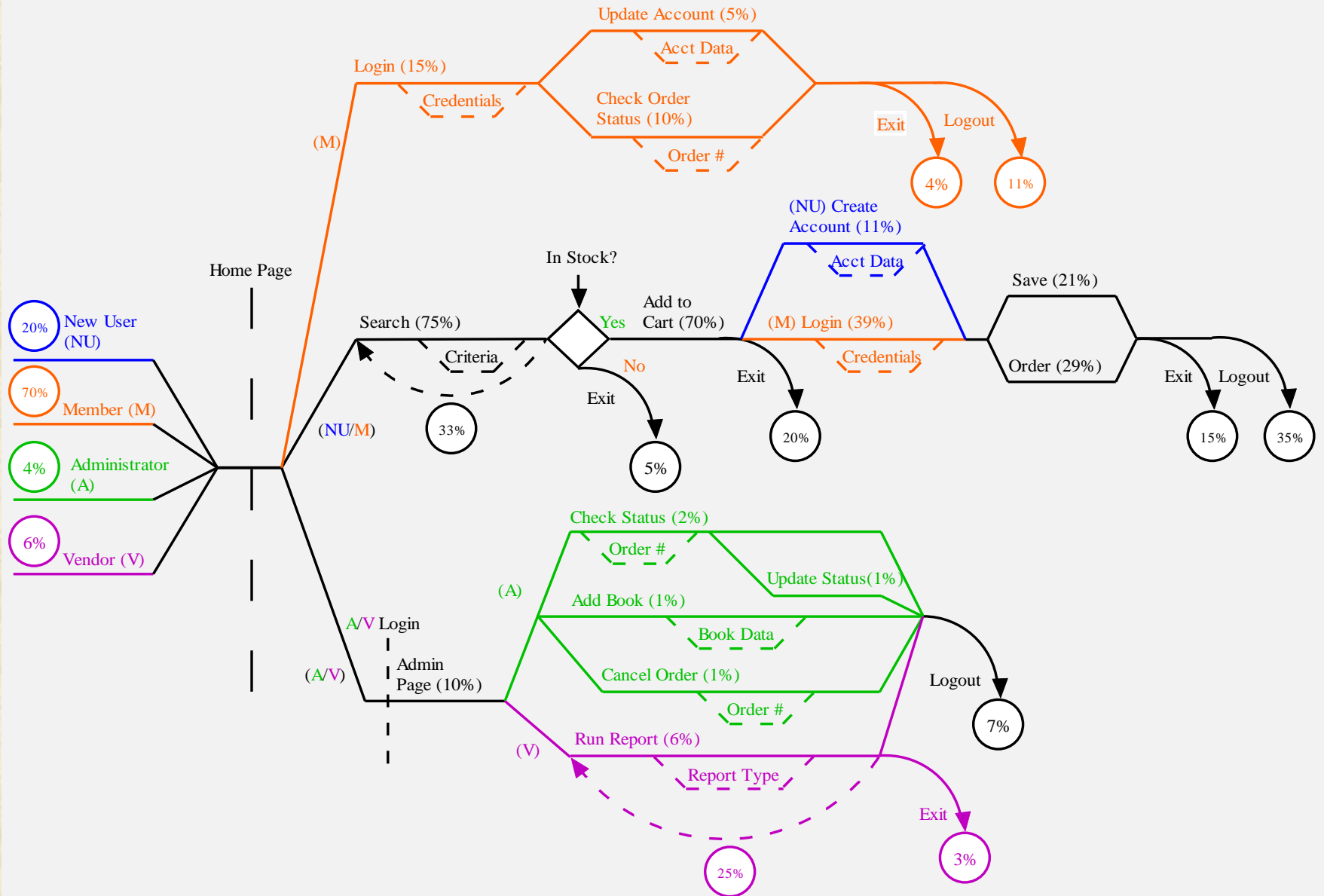
Administrators – Administrate Books and Orders

Vendors – Run Stock, Sales and Order Reports

Other things to think about:

- Navigation Paths
- Conditional Navigation
- Single User Multiple Activities (i.e. Looping)
- Exit and Abandonment situations
- Data Variance

UCML 1.1™ (User Community Modeling Language)



Supplementary Information

Overview

- Model alone doesn't tell the whole story.
- Must also document variances such as:
 - Specific inputs and location of data
 - Details of conditions
 - Logic behind exits
 - Variance patterns and distributions
 - Delays between activities
 - Synchronization Point details
- Model is of limited use without supplementary information (and vice versa).

UCML 1.1™ (User Community Modeling Language)

Activity						
Create Account						
Think Time (sec)	Min	Max	Std	Distribution		
	25.0	60.0	8.0	Normal		
Abandon (sec)	Min	Max	Std	Distribution	Event	
	60.0	120.0	N/A	NegExp	Abandon	
P/F Condition	If Fail, log data and abandon user.					
Acct Data	Field	Variable Name	Data Description		Data Location	
	Ccard	int_ccard	Valid C-card #'s		File.csv	
	Exp_date	int_exp	Valid E-date for C-card		File.csv	
	Name	str_cname	Valid Name for C-card		File.csv	
	Street	str_street	Valid Street for C-card		File.csv	
	City	str_city	Valid City for C-card		File.csv	
	State	str_state	Valid State for C-card		File.csv	
	Zip	str_zip	Valid zip for C-card		File.csv	
Sync Point						
Home Page						
Type	Parameter(s)					
Navigational	None					
Condition						
In Stock?						
Criteria	Resulting Activity(s)					
Yes	Purchase					
No	Exit					

Summary

Value

- Intuitive to all members of the team.
- Replaces complex modeling and documentation methods.

Use

- Visualize and document system usage and parameters.
- Build Test Plans and Scripts directly from the model.

Usage Data Collection – Provides common language for data gathering.

Creation

- Create using common drawing software like MS Visio or SmartDraw.
- Use library of symbols

Supplementary Information

- Use spreadsheet (like Excel) to document associated data.
- Include all data needed for planning, scripting and coverage.

Where to go for more Information

Official specification is maintained at:

<http://www.perftestplus.com>

Examples of use for Performance Workloads:

User Experience, not Metrics and *Beyond Performance Testing* articles on <http://www.perftestplus.com> and <http://www.rational.net>

Version 0.9: documented by Nathan White, a retired military officer currently working for A. G. Edwards , based on *User Experience, not Metrics* and *Beyond Performance Testing* articles articles.

Version 1.0: initially published on the Rational Developers Network <http://www.rational.net>.