Reducing the Cost of User Acceptance Testing with Combinatorial Test Design

Steven Dyson

steven.dyson@cambiahealth.com

Abstract

At Cambia Health Solutions, User Acceptance Testing (UAT) is part of our typical testing strategy to ensure that software solutions fit with the workflow and manual processes for efficient business operations. Our UAT is often a manual process making it expensive, time consuming, and heavily reliant on shared business resources. Long UAT cycles have also led to delayed deployment schedules, which left development teams dissatisfied and held up their work efforts.

Two years ago, Cambia began utilizing Combinatorial Test Design (CTD) to analyze the UAT test cases on some of our larger projects. By analyzing our software parameters and values we were able to consistently optimize our test suites by eliminating duplicate test interactions while maintaining test coverage and realize a reduction in the cost and duration of our UAT cycle. This optimization also improved our predictability during the agile software development life cycle.

This paper details how to incorporate the use of CTD into a testing strategy, how to analyze and parameterize UAT test cases, how to setup and use the CTD application Hexawise, and provides an overview of the impact and results CTD has had at Cambia. Our typical optimization resulted in at least a 30% reduction in required testing time, with as much as a 92% reduction in one test suite.

Biography

Steve Dyson spent 4 years working in User Acceptance, Integration and System testing at Cambia Health Solution before moving into Software Quality Assurance 2 years ago. Recently, Steve has been building an SQA training program at Cambia to standardized SQA, testing and agile concepts and practices within the company.

Steve has a business degree in Accounting from the University of Utah.

1. Combinatorial Test Design (CTD)

1.1. What is Combinatorial Test Design?

Software is always evolving and growing in complexity. Research conducted by the National Institute of Standards and Technology (NIST) suggest that many software failures are caused by the interaction of a relatively small number of variables (NIST, “Combinatorial and Pairwise Testing”). Since large applications could have dozens of different parameters that interact amongst each other, the number of unique interactions could be in the thousands (if not millions). Running thousands of tests is costly and time consuming, and Combinatorial Test Design can help solve that problem.

Combinatorial Test Design, or Pairwise Testing, is a test modeling approach that takes the parameters of a software application and combines the interactions between those parameters into an optimized set of test cases based on a defined level of interaction. All parameters and their corresponding values are combined with one another in a test set. The test set produced contains the full Cartesian product of parameter interactions and consists of the smallest number of possible test cases, while ensuring 100% test coverage based on the defined parameters and values.

1.2. Benefits and Uses of CTD

CTD is a very useful process when testers need to:

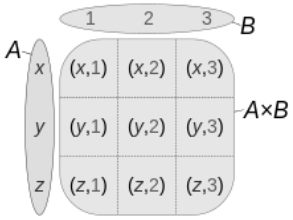
- Create a large test suite from scratch without having to manually write out each test case
- Identify testing gaps in existing test suites
- Eliminate duplications in existing test suites

Test planning can be time consuming, so using the CTD process not only saves us time in our test execution, it also saves time by eliminating the need to manually write extensive test suites.

1.3. How CTD Works

Once the parameters and values for the application have been gathered, it is best to organize them in simple tables. When the parameters and values are entered into a CTD tool, the Cartesian product is created. A Cartesian product is a mathematical operation which returns a single set from multiple sets (Cartesian product, Wikipedia).

Figure 1, Cartesian Product



This simple product of set A multiplied by set B, gives us a new set called AB. Set AB contains every interaction between the values in Set A and Set B, and every interaction is present in our new set. Test cases can be structured in the same manner – a value of one parameter being tested with the values of other parameters.

Of course, software applications are not this simple and can be comprised of dozens of parameters. As we add more parameters, and those parameters contain more than just three values, the Cartesian product grows exponentially.

The CTD tool takes that Cartesian product of every possible test interaction, and combines and condenses those interactions into as few test cases as possible. Duplicated interactions are eliminated, and every interaction is present at least once in the new test suite.

In figure 2, as the Cartesian product for this example is written, notice how many values are duplicated throughout the first six test cases. CTD helps eliminate the redundancy in a test suite, by forming efficiently combined test cases.

Figure 2

Gender	Coverage	Claim Payment Status	Claim Form	Documentation	Claim Type	SSA	Disabled	Working
Male	Primary	Fully Paid	Properly Formatted	Record of Insurance	Professional	SSA	Disabled	Working
Male	Primary	Fully Paid	Properly Formatted	Record of Insurance	Professional	SSA	Disabled	Not Working
Male	Primary	Fully Paid	Properly Formatted	Record of Insurance	Professional	SSA	Not Disabled	Working
Male	Primary	Fully Paid	Properly Formatted	Record of Insurance	Professional	SSA	Not Disabled	Not Working
Male	Primary	Fully Paid	Properly Formatted	Record of Insurance	Professional	No SSA	Not Disabled	Not Working
Male	Primary	Fully Paid	Properly Formatted	Record of Insurance	Professional	No SSA	Not Disabled	Not Working

1.4. Hexawise

Hexawise is a web-based test design tool specializing in CTD. It allows the user to add or import their test attributes, add special test requirements, create combinatorial tests, and add custom test scripts (Hexawise). This tool will be referenced throughout this paper.

1.5. CTD Example

This example is a simplified version of test requirements for Medicare insurance claims in a claims processing system. We have identified the parameters that need to be tested, along with their corresponding values.

Parameters

- Gender of the patient
- Whether Medicare is the patient's primary or secondary insurance policy
- The status on payment of the claim
- Whether the Claim form was submitted properly or not
- If we have prior documentation of the patient's insurance history
- The type of claim being processed
- Whether the patient has Social Security (SSA) or not
- Whether the patient is on Disability or not
- Whether the patient is working or not

For the sake of organization, and import capabilities into CTD tools, writing the parameters and values in this format is very important.

Figure 3, Parameters and Value Format

Gender	Coverage	Claim Payment Status	Claim form	Documentation	Claim type	SSA	Disabled	Working Status
Male	Primary	Fully paid	Properly formatted	Record of insurance	Professional	SSA	Disabled	Working
Female	Secondary	Partially paid	Improperly formatted	No record of insurance	Medical	No SSA	Not Disabled	Not Working
		None paid			Dental			

From this simple example, with each parameter having no more than three values, there are 1,152 possible interactions that can be tested – Male patient tested with Medicare as the primary coverage, Female patient tested with Medicare as the primary coverage, Male patient with a Claim Fully Paid, etc. The Cartesian product of possible interactions grows tremendously as we add more parameters and values to the test plan. Those 1,152 interactions can be combined because each interaction only needs to be tested once. After being run through a CTD tool, all 1,152 interactions can be tested with 10 total cases. Our small test suite provides 100% test coverage of the interaction between any two variables.

Figure 4

Test Number	Gender	Coverage	Claim Payment Status	Claim form	Documentation	Claim types	SSA	Disabled	Working
1	Male	Primary	Fully paid	Properly formatted	Record of insurance	Professional	SSA	Disabled	Working
2	Female	Secondary	Partially paid	Improperly formatted	No record of insurance	Professional	No SSA	Not Disabled	Not Working
3	Female	Primary	Fully paid	Properly formatted	No record of insurance	Medical	No SSA	Disabled	Not Working
4	Male	Secondary	None paid	Improperly formatted	Record of insurance	Professional	SSA	Not Disabled	Working
5	Female	Secondary	Fully paid	Properly formatted	Record of insurance	Dental	No SSA	Not Disabled	Working
6	Male	Primary	Partially paid	Improperly formatted	Record of insurance	Medical	SSA	Disabled	Working
7	Male	Primary	Partially paid	Properly formatted	No record of insurance	Dental	SSA	Disabled	Not Working
8	Male	Secondary	None paid	Properly formatted	No record of insurance	Medical	No SSA	Disabled	Working
9	Female	Primary	None paid	Improperly formatted	Record of insurance	Dental	SSA	Not Disabled	Not Working
10	*Male*	*Primary*	Fully paid	Improperly formatted	*No record of insurance*	Medical	*SSA*	Not Disabled	*Not Working*

Every test scenario is unique, and the amount of duplication of parameter interactions is reduced as much as possible. All of the possible interactions have been combined together saving the tester from exhaustively writing every possible test scenario, which can be time consuming and difficult to accomplish.

2. CTD Process Steps

2.1. Discovery/Test Planning Phase

The discovery phase of the process is when we begin the analysis of the application we need to test. It doesn't differ much from typical test planning, but the train of thought needs to be geared towards parameter identification, and how parameters interact with each other in a system. These activities may include:

- Discuss with a Subject Matter Expert (SME)
- Review process and system documentation
- Review existing test cases and test suites
- Review historic defect reports

If we are leveraging a set of existing test cases, we start by reviewing the test cases and restructuring them in the parameter and value format. These existing test cases in this format can be imported into Hexawise quickly and efficiently.

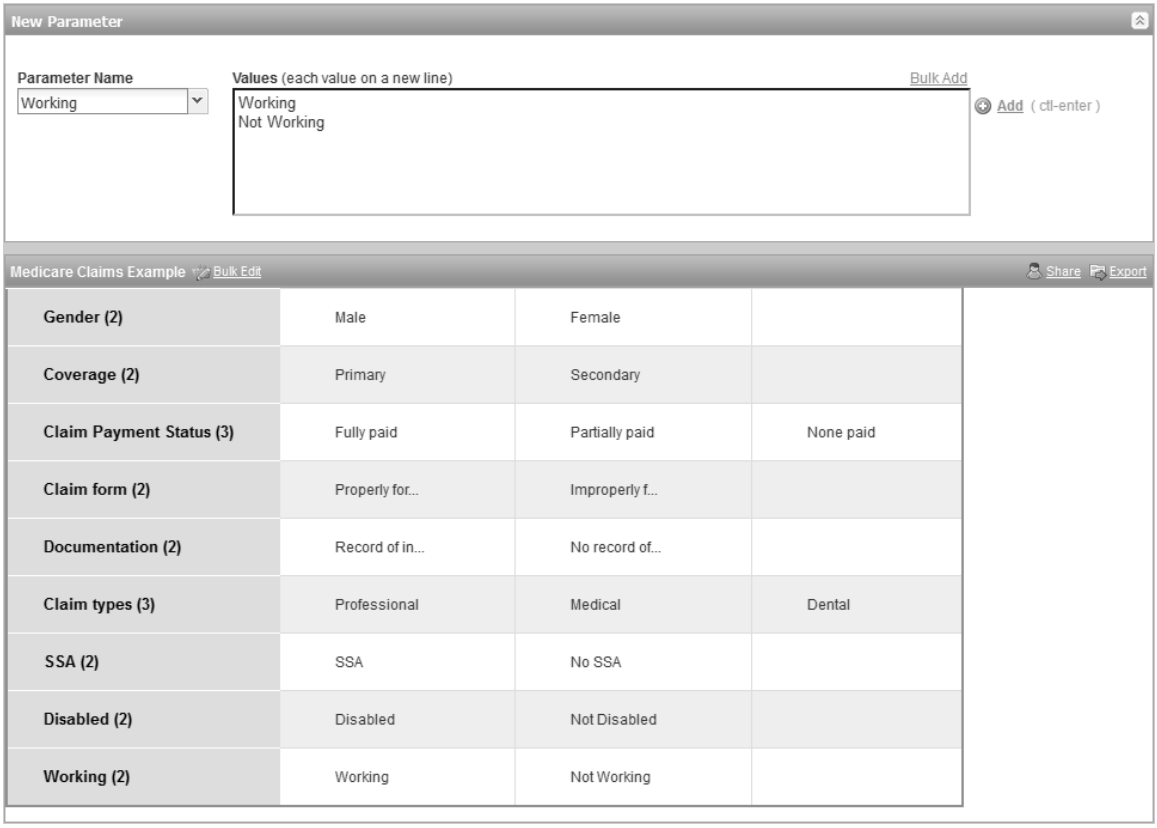
If we are creating a new test suite from scratch, we start by reviewing all process documentation and requirements to ensure we are capturing all parameters in the application. Missing a parameter that

needs to be tested can compromise the new test suite. This is why working with a SME is very important, because their experience and firsthand knowledge of the application may include details that aren't present in documentation.

2.2. Entering Parameters Into Hexawise

Once we have identified the test scope in the application, we compile the test requirements into the parameter and value format. With Hexawise, these can be manually entered or imported from an Excel file.

Figure 5

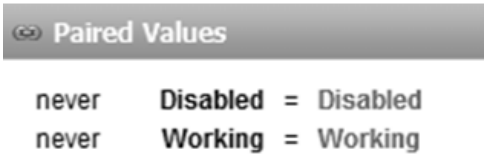


2.3. Restrictions

There will often be interactions in a test plan that we have knowledge can't exist, or wouldn't exist in a test or production environment. We don't want to see these bad interactions muddying up our test plan, so we can restrict them. Restrictions are the interactions that should not be present in the test suite.

If it is known that the value of *Disabled* would never interact with a *Working* value, as seen in Figure 5, that combination can be marked as an Invalid Pair in Hexawise as seen in Figure 6.

Figure 6

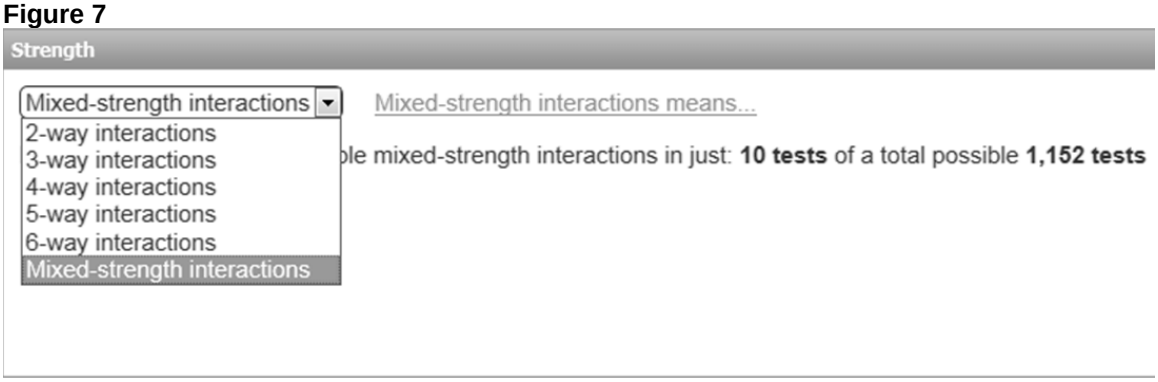


This interaction will then not show up in any of our test cases. We can add as many restrictions as necessary to ensure our test suite contains only good test cases.

2.4. Interaction Levels

Interaction levels in CTD and the Hexawise tool can be defined, based on whether we want value interactions to be matched in pairs of 2 (pairwise testing), sets of 3, etc. As Interactions levels increase the test suite will become more thorough, but the size of our test suite will also increase.

Figure 7



Pairwise testing is more than sufficient for the majority of software testing, but if we're testing software for the International Space Station, we may want to increase the interaction level. Pairwise testing ensures every pair of values is present in the test suite. Not every set of 3 values, 4values, etc. will be present, however.

If an application has stringent requirements, in which 3 or more attributes have specific interactions that need to be tested, it is recommended to increase the interaction strength. This will increase the number of test cases in the test suite to accommodate those requirements.

2.5. Executing Tests with Hexawise

Creating test cases in Hexawise is quick and simple. After we have defined our inputs and any restrictions for bad value pairs, we are ready to create tests. With the click of a button, Hexawise runs all of the value combinations together, excluding any restrictions we have defined, and generates the test suite. As mentioned previously, we can test all interactions with only 10 test cases. Notice that our restriction of the bad pair *Disabled* and *Working* does not appear in our test set.

Figure 8

Strength

2-way interactions | 2-way interactions means...

Hexawise tests cover all possible 2-way interactions in just: **10 tests** of a total possible **1,152 tests**

Medicare Claims Example (2-way interactions) - values in purple *italics* can be replaced with any valid value

	Gender	Coverage	Claim Payment Status	Claim form	Documentation	Claim types	SSA	Disabled	Working
1	Male	Primary	Fully paid	Properly formatted	Record of insurance	Professional	SSA	Disabled	Not Working
2	Female	Secondary	Partially paid	Improperly formatted	No record of insurance	Professional	No SSA	Not Disabled	Working
3	Female	Secondary	Fully paid	Properly formatted	No record of insurance	Medical	No SSA	Disabled	Not Working
4	Male	Primary	None paid	Improperly formatted	Record of insurance	Professional	SSA	Not Disabled	Working
5	Female	Primary	Fully paid	Properly formatted	Record of insurance	Dental	No SSA	Not Disabled	Working
6	Male	Primary	Partially paid	Improperly formatted	Record of insurance	Medical	SSA	Disabled	Not Working
7	Male	Secondary	Partially paid	Properly formatted	No record of insurance	Dental	SSA	Disabled	Not Working
8	Male	Primary	None paid	Properly formatted	No record of insurance	Medical	No SSA	Disabled	Not Working
9	Female	Secondary	None paid	Improperly formatted	Record of insurance	Dental	SSA	Not Disabled	Not Working
10	*Female*	*Primary*	Fully paid	Improperly formatted	*No record of insurance*	Medical	*SSA*	Not Disabled	Working

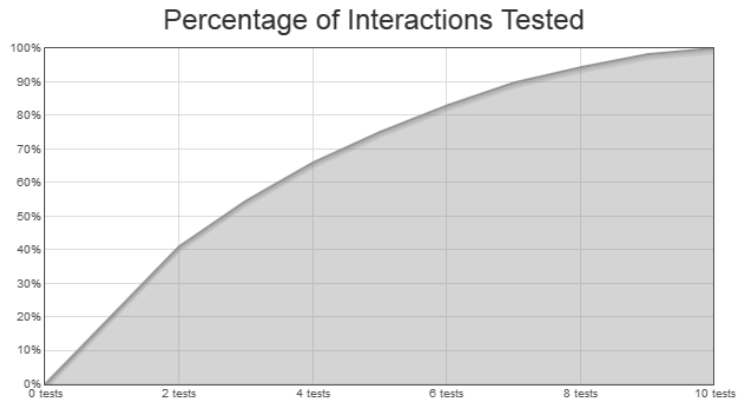
These tests can now be exported to Excel or HP Quality Center. Hexawise provides the user the ability to add auto-scripts if there is a need more specific detail in the test set.

2.6. Test Case Coverage Analysis with Hexawise

One note about a test suite created by CTD is that the numerical order of test cases is important. In our test example, we have test cases 1-10. As we progress through the order of test cases, the number of unique interactions decreases as Hexawise exhausts all possible combinations. This means that if testing time is limited and not all tests can be executed, it is important to test through the numerical order to ensure the highest test coverage possible.

As seen in Figure 9, the Percentage of Interactions Tested indicates that test case 1 and 2 represent 40% of all possible interactions. That is a significant portion of our testable values. As we progress through the test cases, the number of unique interactions tested decreases.

Figure 9



In the event limited testing time is available, this feature of Hexawise is valuable in that it aids the testers in defining an acceptable level of test coverage, and tells them how many tests need to be executed.

3. Cambia's User Acceptance Testing

In 2012, Cambia Health Solutions hired a consulting firm to help identify solutions for finding process efficiencies in our projects in preparation for the federally mandated upgrade to ICD-10 (which is the reclassification of all diagnosis codes used in medical billing). The firm suggested utilizing the CTD method in order to optimize our more complex UAT regression test suites. We began with a pilot to see if the process would work well for us. After adopting the process we optimized many of our existing test suites, and created brand new test suites for new applications.

3.1. Project 1 – CTD Pilot: Policy Benefit Validation

For our pilot run for CTD, a large regression suite was chosen. The regression suite covered insurance policy benefit validation, and was used to test every insurance policy we offer. When a new group signs up for our insurance products, they often have multiple policies. Our regression suite was designed to test a single policy, and the suite contained 667 test cases. On average, it took our user acceptance testers around 40 hours to complete the testing for a new group's entire set of policies. This was far too labor intensive, and we needed to find a way to optimize our test suite.

We met with a SME to identify all of the parameters, and their respective values that needed to be tested. We also identified any restrictions, or interactions between parameters that we did not want present in the test suite.

Seven different parameters were identified:

- Product
- Type of Service
- Provider Category
- Diagnosis
- Procedure
- Benefit Event

Each parameter had various values assigned. For Example, Provider Category contained three values: In Network, Participating, and Out of Network; while the Benefit Event parameter contained values such as deductible, coinsurance, copayment, and yearly visit maximum. The Type of Service parameter contained over 70 different benefits that needed to be tested, which was the driving force of the size of this test suite.

Results

We ran this test suite through the CTD tool and saw great results. We discovered that our initial test suite of 667 test cases was missing 23 test scenarios, which brought us to a total of 690 cases. The test suite that was produced from the CTD process reduced our total number of test cases to 230 – a 67% reduction in the test suite size and testing time. This meant that a tester could now test three groups in the same time it took them to test a single group.

The savings were tremendous, so we made the decision from our pilot to begin utilizing the CTD process for as many UAT test suites as possible.

3.2. Project 2 – Other Party Liability Test Suite Enhancement

CTD doesn't always result in a lower number of test cases, as we found in one of our project examples. I consulted the subject matter expert (SME) on a project team that tested the software which processed insurance claims with members who had a primary and secondary insurance in the event of an accident, also known as Other Party Liability (OPL). The other insurance was typically an auto insurance carrier. She had recently taken over as the user acceptance tester and was leveraging a preexisting test suite that wasn't very thorough.

We took the existing test suite containing 27 test cases, broke it down into the parameter format, and ran the inputs through the CTD tool.

Seven different parameters were identified for the new test suite:

- Group Type
- Provider Type
- OPL Condition
- Additional Claim Needed
- Accident Date
- Second Claim Accident Date
- Body Part Match
- Diagnosis

Results

We quickly found out that our test coverage was terrible, and there were several attributes that weren't being tested. We discovered that our current test suite only had about 30% test coverage, excluding the attributes that were missing.

Our newly expanded test suite now contained 112 test cases, but we had 100% test coverage based on the attributes defined by the new SME. After running the new test suite, we identified two high severity defects that had been in our production environment for months.

3.3. Employee Web Services Project

By far our best result with using CTD was on a web service testing project for an employee insurance website. The tester wanted to test the interaction of fields on web searches, to validate that error codes appeared due to certain combinations or if fields were left blank. The tester had basically mapped out a lot of the Cartesian product of all possible value interactions, so CTD was a perfect fit. There were a total of 228 test cases in the original test suite.

Eight different parameters were identified:

- From Date Search
- To Date Search
- Invoice ID
- Group ID

- Receipt ID
- Matching Invoice
- Matching Receipt
- Matching Group

Results

Since the original test suite was basically the Cartesian product, and CTD's algorithm condenses the Cartesian product, the reduction in test cases was remarkable. Our test suite created by CTD had 100% test coverage based on our defined attributes and values with only 17 test cases, which was a 92% reduction in the number of test cases that needed to be run. The duplication of test interactions in the original test suite contributed to the 228 test cases, so by combining those duplication interactions, we proved that the test suite required the tester to test the same things over and over again.

4. Analysis

Cambia has had a lot of success thus far with CTD and we are hoping to increase adoption within our organization on our various agile development teams. This test planning method didn't work for us in every situation and we spoke to many teams in which CTD just wasn't a good fit. Some of those reasons include a lack of variability in attributes, their test attributes needed to match database rows eliminating the value gained from test combinations, and applications that couldn't be broken into an attribute and value format.

Here are the results from seven projects in which we were able to utilize CTD at Cambia to improve existing UAT test suites (including those discussed in Section 3).

Figure 10

Project	Original Test Case Count	CTD Test Case Count	Results & Impact
A	667	230	Identified 23 missing tests and reduced number of test cases by 65%
B	27	112	Original test suite only had ~30% test coverage of defined test requirements. The increase of test cases closed our testing gaps.
C	228	17	Reduced number of test cases by 92%
D	121	84	Reduced number of test cases by 30%
E	215	120	Reduced number of test cases by 44%
F	61	41	Reduced number of test cases by 33%
G	34	52	Original test suite only had 65% test coverage of defined test requirements. The increase of test cases closed our testing gaps.

Any time we had a reduction in the number of test cases, we saw at least a 30% reduction, and any time we had an increase in the number of test cases, we had a drastic improvement in our test coverage.

5. Conclusion

Combinatorial Test Design is a great test planning method and should be considered in any testing strategy when the software under test is in a parameterized structure. There are several benefits that can be realized with CTD including time savings in test planning and test execution, along with optimizing existing test suites.

The method is not going to meet the entire test planning needs of a team, but it can serve as a valuable supplemental process for systems with a high number of attribute and value interactions. Cambia has been able to realize real financial, and time, savings with this method and we're looking to expand into other areas of our organization.

References

Web Sites:

National Institute of Standards and Technology (NIST). "Combinatorial and Pairwise Testing." - http://csrc.nist.gov/groups/SNS/acts/index.html (accessed July 10th, 2014).

Hexawise. http://www.hexawise.com (accessed June 14th, 2014).

Wikipedia. "Cartesian product". http://en.wikipedia.org/wiki/Cartesian_product (accessed June 14th, 2014).