Red Centre Software
Data Comes Alive

Automating Continuous Tracking:

The Ideal System

Delivered at Association for Survey Computing conference:

Getting the Message Across – Automating and Communication Survey Results

Imperial College, London, October 2008

Version: 25 September 2008

[minor edits March 2009]

[minor edits September 2009]

AUTOMATING CONTINUOUS TRACKING

This document outlines the essential steps for job setup, data processing, and reporting of continuous tracking jobs, emphasing strategies and techniques for maximising the level of automation.

Introduction

Why is continuous tracking (CT) so hard? Since ad hoc jobs are easy enough, why is a CT job not just a series of ad hocs organised by time? Why do things keep going wrong?

Dynamism

Because the only point and purpose of CT is to respond to dynamism – in a static world, a single ad hoc job would be enough for everything. But in a dynamic one, with evolving markets, increasingly disparate trends in consumer behaviour, and an ever-shifting mix of competitors and products, a single frame of the moving picture can be very deceptive. Dynamism in the subject matter of a survey means that the survey instrument itself must evolve over time, or become increasingly irrelevant. Failure to correctly manage the job-wide implications of the evolution of the instrument is the paramount cause of difficulties. In practice, a failure to manage dynamism and evolution (change) is often a combination of poor work practices and inadequate software.

This paper posits the minimal work practices and software functionality required to automate CT jobs as substantially as possible, and seeks to establish a set of underlying principles which can govern decision making for all types of tracking studies.

Managing Change - A New Brand

A new brand comes on the market. The implementation tasks are listed on the left, and the problems which could follow omission or mistakes are on the right.

Task	**Damage Risk**
Edit the master questionnaire and log the change	No metadata makes historical interpretation in later years problematic
Consult the existing brand lists and allocate a code	Reuse an existing code, thereby destroying the data for both Different codes will be used for the same brand across different variables, leading to DP chaos
Brief the interviewers (CATI) or script writers (internet collection)	The wrong brand is asked for, or is incorrectly coded or labeled
Confirm that the field service's collection system will capture and validate the new code	Miss deadlines and incur the client's wrath as you try to sort out the mess No data to assess a campaign start

Identify verbatims where the brand could be mentioned, and brief the coding department accordingly	The new brand will end up merged in with Other or No Further Info – which typically will go unnoticed for weeks. The coders will give the brand an arbitrary code, thereby breaking the integrity of the master brand list and creating DP chaos.
Add the new code and label to the brand lists maintained in the DP house system	Data is collected, but never processed, ending up as Not Established or Undefined
Check all the relevant variables which store and decode the new data item	Source data will become internally inconsistent
Check any constructions which net or otherwise use the brand list	Analysis will become the pursuit of strange anomalies
Consider all table and chart specifications which use brand lists for axes or for filters	Crosstab outputs will grow increasingly awry until an analyst calls them out as absurd (Coca-Cola at 5% share?), usually sometime after the results have already been reported to the client
Connect the new brand to the reporting regime	Connects to the wrong brand label or spreadsheet formula, Client terminates account
Modify diagnostic reports to catch issues arising in any of the above	All of the above, especially *Client terminates account*

The worst case is a new parent brand, because that will require child brand codes to be allocated too, each as per the above steps. The labour is large - hence the risk of oversight, miscommunication, misunderstood instructions etc is high. Things will go wrong – it is only a matter of time, and in my experience, not a lot of it.

Managing Change - A New Bank of Variables

As well as new codes to existing variables, often whole banks of new variables are introduced, typically to assess things like the response to a heavy promotion, images related to a new execution, an expansion into a new or related line of business, unexpected actions by a competitor, or to implement a new measure for consumer behaviour, and so on. For a new variable which categorises the brand list, then in addition to the above implementation tasks and risks, we have

Task	**Risk**
Provide a set of consistent variable names	In lieu of specification, the questionnaire nomenclature will get used

	Variables cannot be referred to collectively, eg BLB_1 to BLB_99, forcing many piece-meal actions instead of a single general one
Decide on the descriptions which will appear in the final reports	QB_12ax etc is meaningless to later users of the data
Make sure that all parent brands have ***exactly*** the ***same*** code across all variables (tautology deliberate)	DP chaos
Make sure that all child brands have ***exactly*** the ***same*** code across all variables	Total DP chaos
Stitch up multi-response variable sets (as per *.SAV) or logically hierarchic data structures to discrete variables	'...lest your variables grow to outnumber the grains of sand or the stars above'
Net variables as required (eg first or other = total, aided or unaided = all)	Analysis will be compromised
Create new tables and charts	Mis-labelled chart series, mis-specified tables
Connect to the reporting regime	Connects to the wrong brand labels or spreadsheets or spreadsheet formulas, Client terminates account
Create a set of diagnostic reports to catch issues arising in any of the above	All of the above, especially *Client terminates account*

Surely all this is obvious? Then why has the stuff in the right hand side happened to me more times than I ever want to think about? Managing the introduction of a new brand code (or any new code) or a new variable is a formal process, yet most companies doing tracking work have never formalised the procedural steps, leaving those charged with actually doing the work to make it up as they go along. With staff stability, eventually by trial and error the procedures will become routine, but where there is a lot of mobility, no sooner are the lessons learned than they are lost again. It can be hard to explain to corporate clients why the same mistakes keep happening over and over again.

In the typical FMCG markets of the major global multinationals (P&G, Unilever, Kraft, Gillette, McDonald's etc), many new brands and variables can occur, sometimes many in a single week. This can be especially true of fast food markets, where the notion of a brand can be extended to a highly promoted menu variation which may be scheduled only for a few weeks. Obviously, the automation of as many of the above steps as possible would have a huge impact on productivity. Where automation is not possible (such as in personal briefings or emailing instructions to field services or accurate coding of verbatims), strict work practices must be observed. In either case, diagnostic routines must be implemented to catch errors at the relevant points in the processing chain.

Heteromorphicity – A Continuous Tracking Job as Shape-Shifter

To accommodate the above procedures for implementing new codes and variables without being swamped by maintenance minutiae and an avalanche of errors a CT job must be by nature heteromorphic – it must routinely and automatically transform itself into different data shapes, where 'data' means a set of categorical or quantitative variables, and 'different' means that both the membership of the set, and the internal structure of the members of the set, can change at any time.

From the IT/DP and processing point of view, this requirement poses many challenges. When I began working in CT for Sutherland Smith and MarketMind in the early 1990s, the process was

Surveycraft tables -> Lotus 123 -> Harvard Graphics

Strangely, now nearly 20 years later, the standard process is

Quantum tables -> Excel -> PowerPoint

Under the work flow procedures above for adding new brands and variables, there are many manual steps involved in getting from the source data processing to table specifications through to spreadsheet formulas and references, and then finally connecting to charts. Often the chart step is manual copy/paste, and often only the most recent data is appended, so any existing errors will never be fixed, and new ones are often introduced. Such systems (and the BI/RDB/OLAP oriented systems even more so) are very brittle in the face of relentless dynamism. They are the antithesis of heteromorphicity. They do not gracefully shape-shift to seamlessly accommodate variations in the structure of the data set – rather, they must be bludgeoned into submission, sometimes fatally.

Error Detection

Furthermore, such systems are opaque against any obviously wrong result. An error at the charting stage can be very difficult to follow back through the spreadsheet manipulations for percentaging and smoothing to the supplying bank of crosstabulations, which are themselves often aggregated from complex networks of inter-related constructions on the source variables. Diagnosis of an output error could take days (and often did). And of course, the worse thought is always that given the obvious errors we did fix, how many subtle ones are present we don't know about? An analyst who reported that Coca-Cola has a market share of 5% just because that is what the chart says would be sacked. But what if the chart says 51% due to a netting error, when the correct value is 49%? The difference is small, and either value is reasonable, but 49% for TCCC would mean that Pepsi had finally got the upper hand in the cola wars – a big psychological victory.

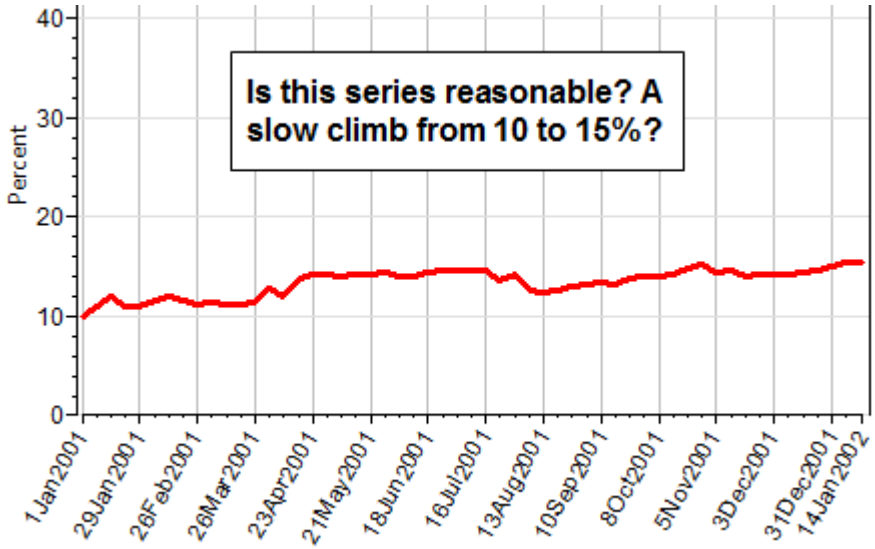
The sad truth is that for an analyst writing a report from just a deck of PowerPoint charts and hard copy tables, there is no way to test the 49% vs 51% proposition – it is taken on trust, and if later found to be incorrect, then hey, we can always blame DP.

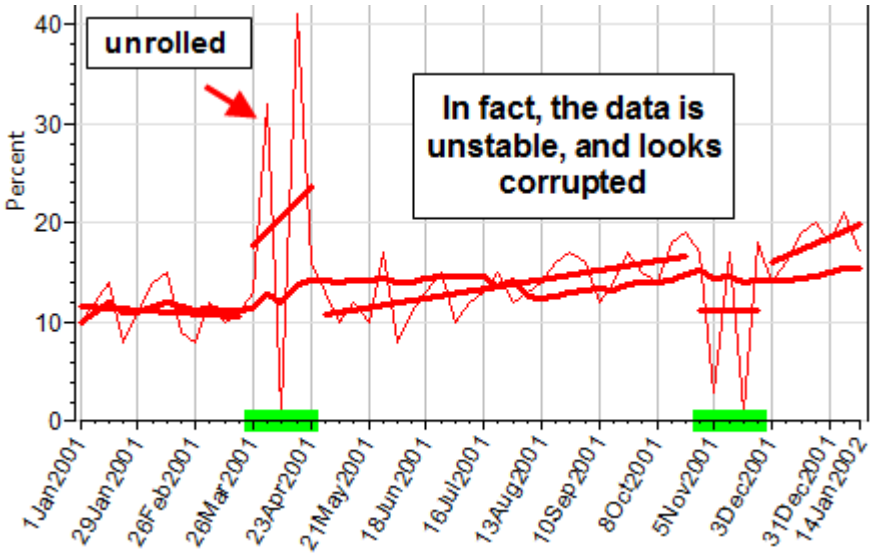
It is better by far to not have any errors, but since continuous tracking and heteromorphism are essentially interchangable concepts, at every update there is

potential for errors to creep in. Good work practices, thorough procedures, and comprehensive diagnostics alone are not enough. Ultimately, an analyst who understands the market must determine that either all results are within expectations, or if not, then that the data trail is solid and that surprise results can be defended with confidence. There is no point whatsoever in hypothesising about the market or consumers if the reason is a DP or field error.

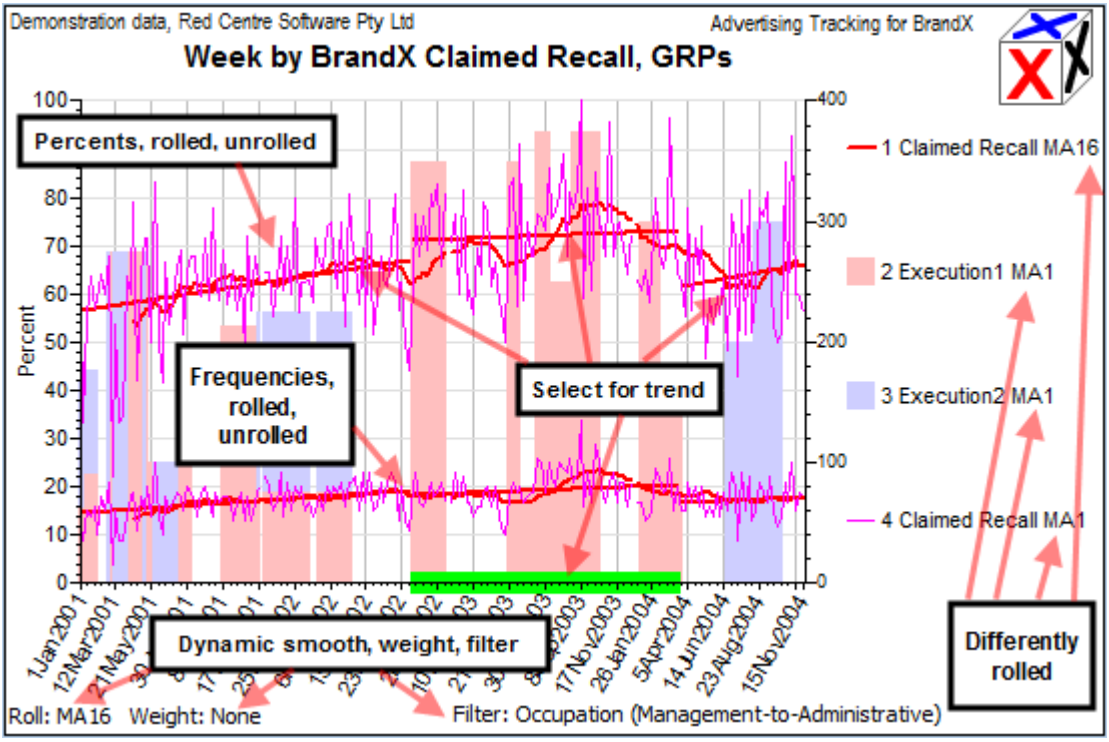
Interactivity

This means that the requirements for an ideal system of automated CT include, paradoxically, full analytical interactivity. The responsible analyst/researchers must be able, at a minimum, to remove/restore percentaging (1/2=50%, and 2/3=66.6%, but both percentages are analytically meaningless), smooth/unsmooth at different levels, unweight/reweight, filter/unfilter, test changes in series behaviour with piece-wise linear regressions, and then, if there is a problem or some uncertainty, unravel constructed variables, examine case data down to the source level and follow from first principles exactly how each final table cell or plotted point was derived. If this cannot be achieved then quality control will remain forever a fanciful and elusive concept. Wrong results will inevitably be reported.





A more complete example:



Achieving Automated Throughput – the Bare Minimum

Achieving automated throughput is largely a matter of building on correct work practices with appropriate software functionality. A minimal account is outlined below. There is quite a bit more, not covered here, which would be essential in any practical sense for heavy duty jobs.

The Principles

There are several principles which guide the underlying rationale. They are

1. Maximum Generality

Organise the job so that as many items as possible can be referred to as simply as possible. A trivial example: referring to codes 1 to 10 is better done as 1/10 than as 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 – ranges, not lists The more general the specifications, the less text, and the less text, the fewer opportunities for error.

In terms of variables, this principle means that you should always work at the highest levels of data abstraction – eg hierarchic and multi-response variables, not atomised single response.

2. Immutable Definitions

Once a code is defined, or a variable named and described, it must never change. Reports use codes in filters, constructions use codes in mappings, and so on. A change of 1=Coke to 1=Pepsi could destroy the job completely.

3. Absolute Consistency

Always use the same code for the same brand. Never do the same thing in different ways. Thinking creatively is fine in terms of job setup, systems and procedures, but once up and running, no variation should ever be admitted. If you want to change a procedure, in view of the risk of messing with the current jobs, it is often better to wait until the next new job comes along.

4. Total Retention

Keep all historical case data and all meta-data well organised and accessible. Ideally, every single case from job inception, and all current and retired or suspended variables, should be immediately accessible. Clients often want comparative analyses, sometimes against very old campaigns, expressly to see how far they may have come over the preceding years. Relying on IT to retrieve a job as constituted five years ago is a dicey proposition. If the job is truly too huge and must be split, then do it yourself using ordinary archiving technology.

5. Don't Bloat

The consequence of the principle of Total Retention is that jobs get bigger and bigger, and can never get smaller. Therefore, never add variables or files to the job without an explicit point and purpose. Always consider whether it is better to construct on the fly as part of a report specification. Only explicitly construct if usage will be frequent.

6. Super-Actions Always

Never do the same thing many times, because that enforces a linear commitment of time and resources. An example would be filtering a set of reports – it is better by far to specify a single global filter which applies to them all, than to be adding the same filter expression to each report, one at a time.

Similarly, updating a PowerPoint deck should be an automatic process, initiated once in constant time, regardless of how many slides or embedded reports.

Adding a new brand to many variables should be done once only and in one place only.

7. The Benito Principle

Organisational Fascism is essential. Military precision and discipline. No exceptions. Mindlessly follow the procedures. Never break the rules. Never take a short cut. Never put a bandaid on a data problem at the end of the chain (like direct edits of a PowerPoint data sheet). Document everything. Keep scrupulous records. Make all individuals in the chain personally accountable for the quality of the data handed on to the next step. Avoid at all costs diffused responsibilities. Demarcate exactly. When things go wrong accountability is paramount – otherwise the problem will not be properly fixed, and will probably soon recur. The earlier in the chain a problem is found and fixed, the less damage will be done.

8. Hyper-defensivity

Be totally paranoid. Think hyper-defensive. The attitude and expectation must be not just 'what can go wrong, will go wrong', but rather 'what can go wrong, *has already gone wrong*'. You just don't know about it yet.

These principles are referred to as appropriate below.

General Work Practices

Document the Job

A poorly documented job is a disaster waiting to happen.

Maintain a Variable Map, Variable Names

The first step to setting up a smoothly running CT job is make sure that the source variables are sufficient to deliver the required measures, that the variable names conform to house standards, that brand and other abbreviations are locked down, that standard descriptions for variables are supplied, and that there is a system for report naming. This can all be done with a single document which I think of as the variable map, because it maps out all the connections between the source variables, constructed variables and reports. A typical sample covering just the standard brand and advertising awareness measures (where AnyBrandX is a master list of all the variant codes for BrandX, ditto AnyBrandY etc, see next subsection) is

Name	Method	Details	Q'nr	Description	Report stem
Unaided Brand Awareness					
TMBA	source		Q1a	Top of Mind Brand Awareness	TopMindBrandAwa
TMBAN	Code net	TMBA(AnyBrandX)=c1... TMBA(AnyBrandY)=c2...		Net Top of Mind Brand Awareness	TopMindBrandAwaNet
UOBA	source		Q1b	Unaided Other Brand Awareness	UnaidOthBrandAwa
UOBAN	Code net	From UOBA, as per TMBAN, UOBA&!TMBAN		Net Unaided Other Brand Awareness	UnaidOthBrandAwaNet
UBA	Var net	TMBA | UOBA		Unaided Brand Awareness	UnaidBrandAwa
UBAN	Var net	TMBAN | UOBAN		Net Unaided Brand Awareness	UnaidBrandAwaNet
Unaided Ad Awareness					
TMAA	source		Q2a	Top of Mind Ad Awareness	TopMindAdAwa
TMAAN	Code net	TMAA(AnyBrandX)=c1...		Net Top of Mind Ad Awareness	TopMindAdAwaNet
UOAA	source		Q2b	Unaided Other Ad Awareness	UnaidOthAdAwa
UOAAN	Code net	From UOAA, as per TMAAN, UOAA&!TMAAN		Net Unaided Other Ad Awareness	UnaidOthAdAwaNet
UAA	Var net	TMAA | UOAA		Unaided Ad Awareness	UnaidAdAwa
UAAN	Var net	TMAAN | UOAAN		Net Unaided Ad Awareness	UnaidAdAwaNet
Aided Brand Awareness					
ABA	source		Q9a	Aided Brand Awareness	AidBrandAwa
Aided Ad Awareness					
AAA	source		Q9b	Aided Ad Awareness	AidAdAwa
Awareness					
UA	Var net	UBA | UAA		Unaided Awareness	UnaidAwa
UAN	Code net	UA(AnyBrandX)=c1...		Net Unaided Awareness	UnaidAwaNet
AA	Var net	ABA | AAA		Aided Awareness	AidAwa
A	Var net	UAN | AA		Awareness	Awa

The variable map must attain a god-like status. Under the Benito Principle, failure to follow it exactly should incur a severe reprimand. Otherwise, you end up with a great jumbled mess of individual preferences for names, abbreviations and descriptions, and no reference point to determine that the constructions have been correctly specified and implemented. That sort of thing is a fertile breeding ground for errors, and makes job navigation difficult for analysts.

A well-maintained variable map can do the job of many other documents, so it should entail less work overall. A date field for new variables would handle the change log function. The details column instructs DP on how to construct. The Report column tells analysts how to name the charts and tables. You can always add to the stem according to other conventions, such as UnaidedAwa_males, or AidedAdAwa_kraft. Together with

an additional document to specify banners and weightings, the variable map can remove the need for a dedicated 'tab plan' or similar.

You can exercise creativity in devising your own conventions and naming schemes and the layout of the map, but once determined and agreed, the map must be followed exactly – no exceptions, not ever, for any reason.

Note that the six source variables (Q1a,b, Q2a,b, Q9a,b) create twelve constructed variables. Two to one is a typical construction:source ratio. It is always important to keep the number of constructions under control. A CT job can never shed a variable, so adding new ones should always be an agreed necessity among the users of the job. Otherwise, over time you will end up with a lot of useless baggage.

Note that my naming convention reduces the name width as more respondents are covered at the greater levels of generality, leading finally to just the variable A, which is the net of all the awareness variables.

In my experience, systems which use named variables, as opposed to fixed-width column references, are best for CT. A variable circumscribes and packages its own data, whereas column references are meaningless on their own, and the columns often shift around. For CT, the flexibility to call any subset of columns anything you like is not necessarily a positive.

Brand Lists and Dynamic Code Frames

Allocation of codes should be done from a position of knowledge about the job and its composite variables, and from a position of sufficient authority to ensure compliance. A well maintained set of brand variables will all share exactly the same code frame, and the code frame itself should be designed with some understanding of the natural groupings. In a worst case scenario of thousands of variant brands across several parents, and with new variants appearing regularly, it is sensible to group the parents in blocks so that they can all be picked up by a single range expression, leaving plenty of room for the years ahead. There is an infinity of integer codes, so there is no reason to be squeezed for space. A good initial arrangement for such a market would be

Parent Brands	Variant Brands
Kraft=1	Kraft=1001/1999
Unilever=2	Unilever=2001/2999
Heinz=3	Heinz=3001/3999

The variants breakdown in the full list would be something like

```
1001=(Kraft) Cheddar Cheese
1002=(Kraft) Vegemite
...
2001=(Unilever) I Can't Believe it's not Butter
2002=(Unilever) Ben and Jerry's
...
3001=(Heinz) Baked Beans (tomato sauce)
3002=(Heinz) Bean Samosa
...
```

Note that codes 1/1000, 2000 and 3000 are sacrificed. This is so that the brand codes always take four digits (keeping banks of filters much better aligned for readability), so that the parent and the variant brands can always have the same leading digit ('1'=Kraft, '2'=Unilever...), and so that boundary problems are avoided (the 'first' is always a trailing '1', never a '0', eg 1001, and not 1000).

Now, at the commencement of the job there might only be 100 or so variants defined for each parent, but if all syntax expressions which need to pick up all parent codes always use the full range, then an important aspect of heteromophicity is implicitly accommodated. A filter expression like UBA=1001/1999 will be true regardless of how many variants have been defined, at least until such time as the number of Kraft variant brands in the market category exceeds 999 (in which case, see Named Code Lists below).

At all costs, under the principle of Absolute Consistency, you must avoid mismatches between codes and brands like this (first instance in bold):

UBA – Unaided Brand Awareness

```
Kraft variant A = 1
Kraft variant B = 2
Kraft variant C = 3
```

UAA – Unaided Ad Awareness

```
Kraft variant A = 3
Kraft variant B = 4
Kraft variant C = 1
Heinz variant A = 2
```

On these code frames, a construction for Unaided Awareness would have to be

UA – Unaided Awareness

```
1 = Kraft variant A = (UBA=1 OR UAA=3)
2 = Kraft variant B = (UBA=2 OR UAA=4)
3 = Kraft variant C = (UBA=3 OR UAA=1)
```

Matching up disparate code frames like this across hundreds or even thousands of codes and variables is a nightmare for DP, and is a fertile breeding ground for outrageous errors. Once allocated and defined, under the principle of Immutable Definitions a code is sacred and inviolate against any change, so it is a good idea to get it right from the outset. In a CT job a poor decision, once entrenched, is often practically impossible to undo.

Log Each Update Step

Every update should be exhaustively documented for the existing number of cases, cases in this update, period covered by this update, new codes and variables, any problems encountered and the actions take to address, source data file names, backups, etc. Without such a log, determining when something went wrong can be needlessly difficult or even impossible.

General Software and System Requirements

If your house system for DP and analysis does not support the following minimum features and functionality, then it is suitable only for the simplest types of static CT.

Use Referencable Master Brand Lists

A typical FMCG job could have twenty variables which use the master variant brand list, and another twenty which use the parent brand list. That is a lot of places to add a new brand code. To automate this you need a software system which can reference master lists. In Quantum, you can use a #include. In Surveycraft, you can cite the full code frame at the first instance, and then refer to that instance (or parts of it) for all subsequent occasions. Other systems have other ways. Find out what yours is, and enforce its use. Updating the master is a Super-Action.

Create Named Code Lists

A master brand list is often just the starting point, however. In a market like disposable razors there are many intersecting segments among the brands. There are single blade, double, triple and triple plus blades, system handles, total throw-away, self-lubricating, male and female specialisations, etc, and these segments cut across all the four parent brands (Gillette, Bic, Wilkinson Sword, Schick). In a market with literally thousands of variants, the only way to manage this sort of confusion is to be able to name the code lists which define each category type, for example, you could have

```
AnySingleBladeDisp=1/10,51/78,102,108,234/300,345,378,401,423,456,502
AnyDoubleBladeDisp=11/22,45,49/50,82/100,111/145
AnyTriplePlusBladeDisp=146/200,301/344,380/400
AnyDisposable=AnySingleBladeDisp|AnyDoubleBladeDisp|AnyTriplePlusBladeDisp
AnySystemHandle=350/360,402/422,550/600
```

These example lists are actually quite a lot shorter than the reality I recall.

Now, if you need to run a set of tables filtered to those who last bought disposable brands, the filter can be trivially specified as BBL=AnyDisposable, and similarly for brand bought ever, as BBE(AnyDisposable), etc. This can avoid a very large number of constructions, and also has the additional advantages of being self-documenting, and of allowing the addition of new items in one place only (the code list) which is then picked up by an indefinite number of instances. To equivalently filter otherwise could require the 20 variables with brand list * 5 product types = 100 constructed variables.

Failure to use master code lists means that at each instance analysts will be making inconsistent decisions about what constitutes a system versus disposable, etc. In Surveycraft, this is an L spec. Other systems have other ways, such as macro definitions. Code lists satisfy the principles of Absolute Consistency and Super-Actions Always.

Code lists can also be used to keep brand nets organised.

```
AnyKraft = 1001/1999,5001,5023,5234
If UBA(AnyKraft) then...
If UAA(AnyKraft) then...
...
```

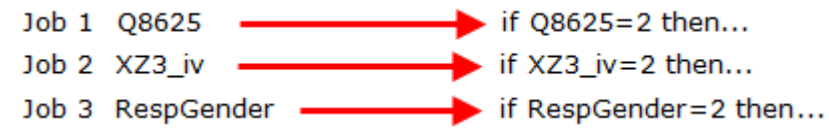
Thus, no matter how many times the Kraft variants are evaluated in expressions, a new variant can be incorporated across the board by simply editing the definition of *AnyKraft* once in some central location.

Map All Jobs to a Consistent Job Structure for Common Variables

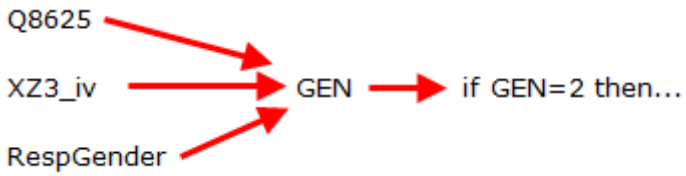
Different field systems have different ways of naming variables, and often use different data structures for storing the case responses within a variable. Surveycraft forces names to be of the form Qxxx where 1<=xxx<=9999. Traditional SPSS has programmatic variable names (leading alpha, alphnumerics, underscore) with an 8 character limit. Some systems do not name variables at all, using the description only, or even worse, just column references. Some systems support high level data structures such as multi-response coded/uncoded hierarchic. Some, such as SPSS, force everything to be low level atomised single response. Under the principle of Absolute Consistency, if your CT jobs require different field services using different data collection systems, with the result that conceptually equivalent variables are routinely delivered under a variety of different names or non-names and are packaged in different ways in different data structures, then having a way of mapping them all to a single consistent set of variables under a single consistent and meaningful nomenclature can avoid a lot of strife.

Under the Don't Bloat principle, any source variable which already structures the data the way you want should be simply aliased to a standard name. For example, if the standard name for a demographic such as gender is GEN, but your Surveycraft supplier is delivering Q8652, your SPSS supplier is delivering XZ3_iv, and your SSS supplier is delivering RespGender, then without aliasing, all of the standard processing for demographics will need to be done in three conceptually equivalent but physically different ways.

Q21c. What is your gender?



Under the principles of Absolute Consistency, Maximum Generality and Super-Actions Always, then by aliasing the many can become one, and all related syntax can instead be uniformly expressed.



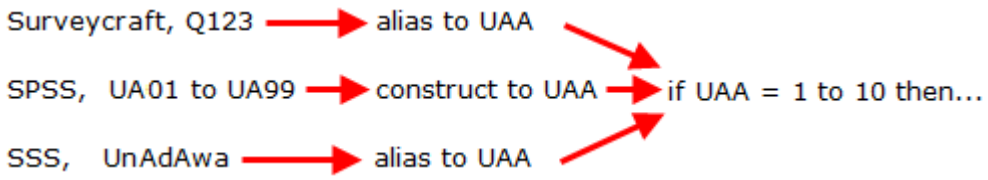
Arranging for maximum commonality across a suite of jobs can dramatically reduce setup and running costs, and hugely improve the navigatability among jobs for analysts. If TMBA always means Top of Mind Brand Awareness, and analysts are assured that this measure is uniformly defined and implemented across all jobs, then job learnings are easily transferred to new jobs, and will rapidly accumulate to expertise and confidence.

Aliasing source variables avoids double storage, but if aliasing is not achievable then consider a batch process or a script to copy/rename. As long as the links are documented by virtue of the process, then the renaming should be safe. Disk storage is a lot cheaper than the time wasted in endlessly respecifying the same logical relationships, and the opportunities for errors arising from mis-specification are much reduced.

Data Structures

A related issue is the membership count and member structure of the variable set. Some systems atomise multi-response (such as SPSS *.sav) and few support hierarchic data in any meaningful way. So if your jobs are sourced from several different systems which employ different data structures to store the case responses, then you will need to map them all to a single set of data structures. Here, the renaming can be done as the constructed target. All mappings will be different, so aliasing saves nothing. For example, for Unaided Ad Awareness, standard name UAA,

Questionnaire text: Q1a. Do you recall seeing any recent advertising for soft drinks or carbonated beverages? What brands were being advertised?



Surveycraft and SSS support multi-response, so aliasing avoids duplicate storage, but SPSS *.SAV requires 99 variables, one for each possible response, so a variable net needs to be applied to reduce the 99 single response variables to a single multi-response variable. Naming the multi-response target UAA at the same time satisifies the principle of Absolute Consistency.

Specification Generators

A system for specification generation is essential for maintaining constructions. In Surveycraft the spec generator operator is < >, so for example <1/10> delivers 1, 2, 3, ...10 in sequence in any context. This sort of thing is essential to heteromorphicity because it allows constructions to be specified in a way which is dynamically self-maintaining. For example, consider the variable net of TMBA with UOBA to get a total UBA. Regardless of the system in use, somewhere a set of conditional filters is evaluated to map the net of codes to UBA. If the rules have been followed, and TMBA and UBA have exactly the same codes, then on a code frame of 1 to 10

```
if TMBA=1 or UOBA=1 then set UBA=1
if TMBA=2 or UOBA=2 then set UBA=2
if TMBA=3 or UOBA=3 then set UBA=3
. . . . . .
if TMBA=10 or UOBA=10 then set UBA=10
```

This is easy enough, but what happens when there is a new code 11? The old way is to manually add another filter

```
if TMBA=11 or UOBA=11 then set UBA=11
```

A spec generator approach could be abstracted as

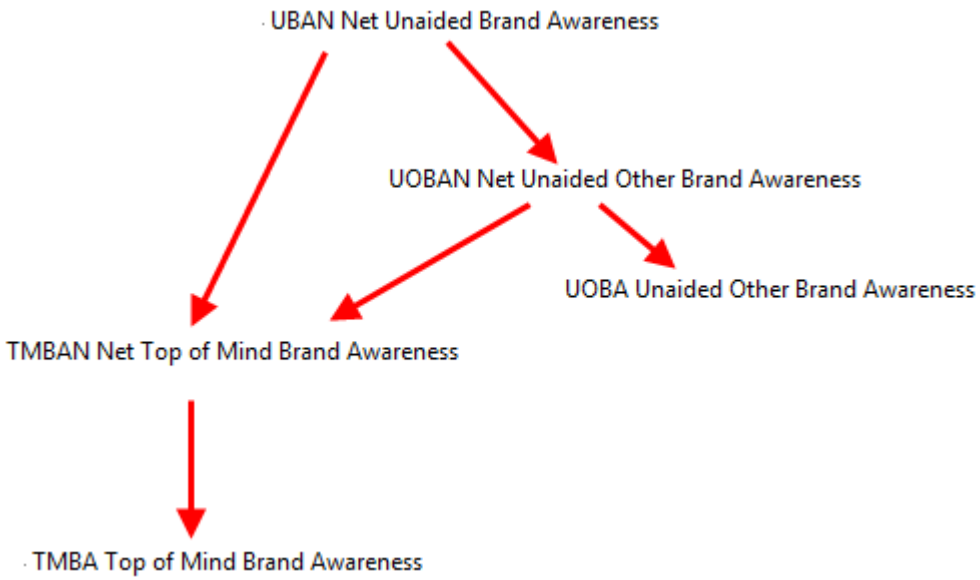
```
if TMBA=* or UOBA=* then set UBA=*
```

where * means 'all currently defined codes for this variable'. This way, it never matters what happens at the source end – the processing system will seamlessly adapt, and the new brand will automatically appear in all constructions which net the brand list. This satisfies the principles of Maximum Generality and Super-Actions Always.

If your system does not support specification generators, then consider writing a parameterised VB script to auto-edit the appropriate files.

Disassembly of Constructions (Variable Ancestry)

The ancestry for UBAN, Net Unaided Brand Awareness, is



These relationships are implicit in the variable map, but can be hard to unravel explicitly. Ancestry trees can get very intricate, so analysts need a way to confirm that the tree is built up in the correct way. If Awareness for Coca-Cola is 5%, which cannot be correct, then where did the problem arise? Ideally, your software should be able to deconstruct the ancestry for any derived variable, and show the logic of the connections – otherwise, a great deal of time will be wasted trying to work out where things went wrong.

See the section *Heteromorphicity in Practice* below for a more complex example.

Rollback

When it is all messed up, you must be able to rollback, reimport and reprocess. Under the principle of Super-Actions Always this should require of the operator only the time it takes to initiate each action. A subtle point is that only the data is affected – all manual extensions to construction definitions or report specifications in this update must remain intact, so a roll-back is not a strip back. Any manually extended structures or reports simply await a fresh pass of the corrected data.

Scripting

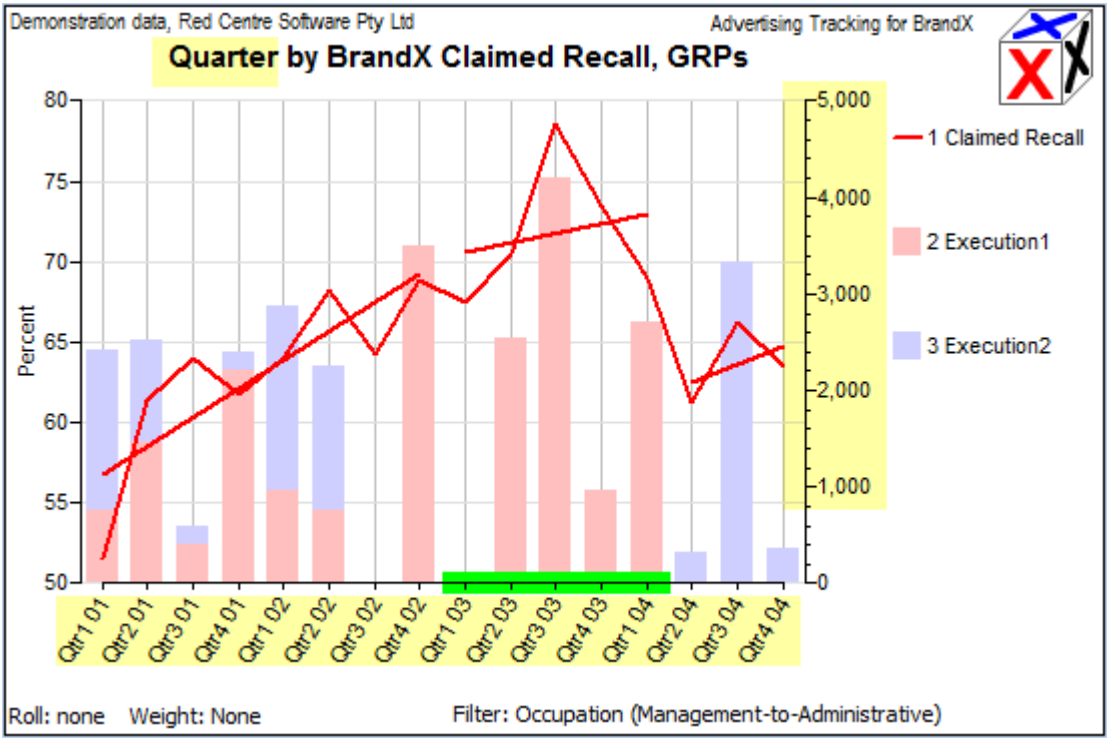
For every manual task which can conceivably be scripted, script it. As soon as possible. Otherwise task time T over N updates will require N*T of the operator's time over the life of the job. If you have done it twice already, then script it now.

Weight within Periods

Otherwise, past data changes as more cases are input to the weighting algorithm, which can be upsetting for clients.

Support External Data

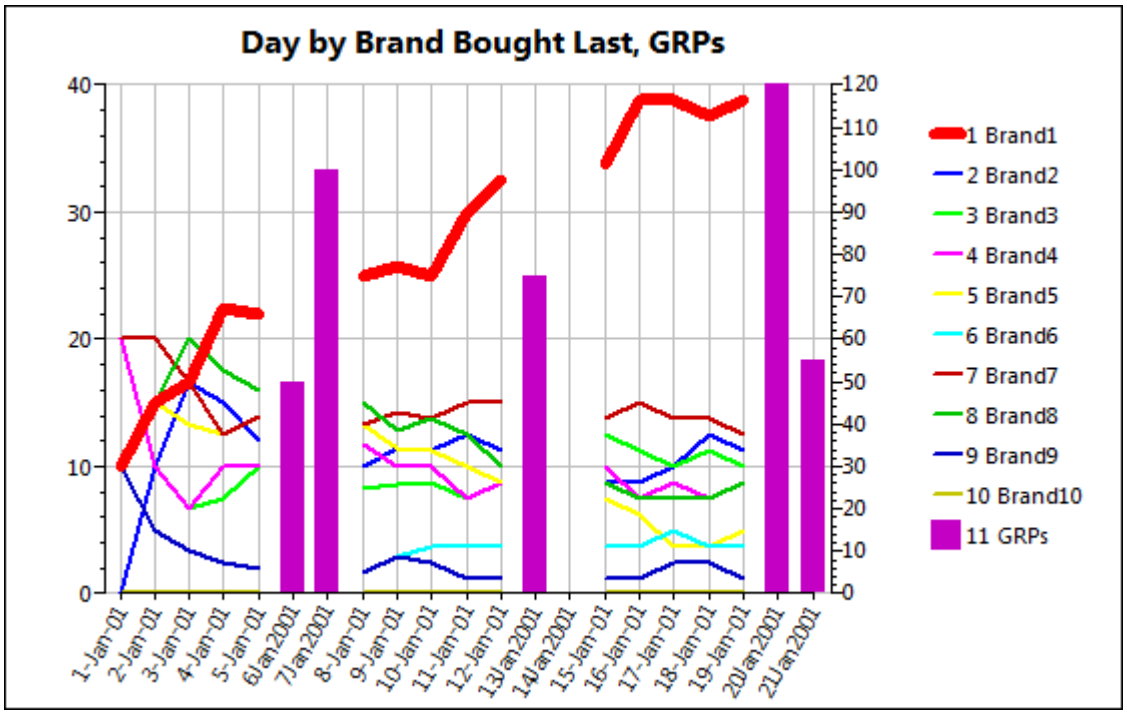
Inserting a row of constants for things like GRPs, TARPs, warehouse withdrawals, CPI index, etc is not generally a problem if working at the same resolution as the data was collected (week for week, month for month, etc). But for the ideal system, analysts need to be able to interactively and seamlessly re-aggregate and/or re-average external data at different resolutions along with survey series.



The principle of Maximum Generality means that analysts should never be locked into only weekly or only monthly windows. In the above chart, the GRP series which is daily external (not survey) data, has reaggregated seamlessly from a Y2 range of 0 to 400 to 0 to 5,000.

True Calendar

Charts especially need to be able to show the calendar on the X axis, regardless of out of field periods. Data collection must be at daily resolution (not weekly) or calendar months cannot be supported, making matching survey series to external monthly indices approximate. Also, competitors may advertise during an out of field period, so GRPs may need to be shown on charts for weeks when there is no survey data.



Here, the survey (at daily resolution) is out of field over the weekends (no day codes are defined for the 6th, 7th, 13th, 14th, and 20th, 21st) , which happens to be just when Brand1 advertises. The rise in Brand1 makes sense when viewed against the media weight.

Diagnostics

No matter how slick and quick the processing, if the final reports are wrong it is all a great waste of time. Maximum damage ensues from errors which persist through to the reporting instrument, requiring a complete roll-back and reprocess to fix. Therefore the sooner an error is caught and addressed the better.

There are six primary techniques:

1. Auto-generate an audit report on all changes to the job at each update
2. Look for and implement as many check-sum relationships as possible
3. Create a table which shows the total for each variable across the last N prior periods and look for unexpected empties or wildly out of character values
4. Create a table specification which compares the update period to N prior periods, and then sort by greatest difference
5. Create charts which obviously visually announce a fundamental problem
6. Run a summary process which reports on all variables for number of cases, high and low bounds for coded, maximum and minimum for uncoded, etc.

For point 4, the greatest differences will be either analytically interesting (poor man's data mining) or indicate a data issue, either of which is well worth knowing. A data issue may be as simple as a variable being rotated out as scheduled, or as fundamental as a miscode in the field.

1. Audit Updates

At each update, a report on all changes with respect to the prior state of the job is essential. This must detail all new codes to existing variables, any changes to the decode labels for existing codes, any changes to the descriptions for existing variables, and all new variables. (Dropped variables and codes are covered by 3) and 4) below.) An example of the sort of output needed is

```
new code TMBA(239), Brand Alpha
new code UOBA(239), Brand Alpha
new code TMAA(293), Brand Alpha
new code UOAA(239), Brand Alpha
changed code def TMBA(238),  old: BrandGamma
                             new: Brand Gamma
changed code def IMAGE1(99), old: Don't Know
                             new: Planet Zeta
```

In a real job, these reports can be very large. Here, there are two problems: TMAA has Brand Alpha as 293 instead of as elswhere 239, and the meaning of IMAGE1 code 99 has changed from Don't Know to Planet Zeta. Both of these are serious issues, and must be explained and addressed. The change of *BrandGamma* to *Brand Gamma* (with a space) is a tidy-up, not an error, but it is always nice to see a fix surface, or to be assured that someone cares, or to check that an instruction has been followed.

2. Check Sum Reports

Many jobs have all sorts of implicit check sums which can be used to test correctness in many areas. Typical circumstances are to check that verbatim coding has been done correctly, or that various nets have happened correctly.

One such simple check sum is UBA = TMBA+UOBA (unaided brand awareness = top of mind plus unaided other). If this does not hold, then the separation into first and other has not been done correctly. Consider this verbatim string in response to an unaided brand awareness question where the only brands are Brand Alpha=1, Brand Beta=2, Brand Gamma=3 and Indeterminate=99:

alpha, love that affy stuff, betamax, gammyleg, alphabet, alf garnet

This then codes as 1, 1, 2, 3, 99, 1. The correct coding would be TMBA=1 and UOBA=2,3,99. It is however very common to see things like UOBA=1,2,3,99,1 - expecially when the respondent has given 20 brands and five of the twenty are spelling variations or synonyms of the same brand. This sort of thing can be easy for human coders under deadline pressures to miss. (Will natural language AI ever rise to the challenge?)

If your system can support it, consider auto-coding all verbatims word for word, and then running tables of auto-coded by human-coded. This can be very useful in catching a change in coding habits at the fringes, most often caused by personnel shifts. A stringent coder might reject *alfgarnet*, and a loose coder might assign *alphabet* as both code 1 and code 2. Under the principle of Absolute Consistency, monitoring the verbatims somehow or other is a pre-emptive necessity – do it manually if there is no other way. Given the sensitivity of many of the measures which derive from verbatims, a coding inconsistency can have dire consequences for analysis.

Reformulating as UBA – TMBA – UOBA = 0, tabulating this expression against Case, and then sorting in ascending order identifies the relevant case IDs, which can then be directly examined. For each of the three example cases, UBA-TMBA-UOBA = 5-1-5 = -1.

TMBA UOBA Overlap Check
Sorted ascending

Case	Frequencies	UBA - TMBA - UOBA
	9955	-1
	9975	-1
	9983	-1
	1	0
	2	0
	3	0
	4	0
	5	0
	6	0
	7	0
	8	0
	9	0
	10	0

Case	UBA	TMBA	UOBA
9952	2;3;4;5	2	3;4;5
9953	1;2	1	2
9954	1;3;5;10	1	3;5;10
9955	1;5;6;8;10	10	1;5;6;8;10
9956	2;4;10	4	2;10
9957	1;4;5	1	4;5
9958	3;5;6	3	5;6
9972	2;5;7	5	2;7
9973	1;3;4;6	4	1;3;6
9974	1;3;4;8	1	3;4;8
9975	1;3;6;7;10	6	1;3;6;7;10
9976	1;2;4;6	1	2;4;6
9977	1;2;3	1	2;3
9978	4;5;7	5	4;7
9980	1;2	1	2
9981	2;3;6;9	2	3;6;9
9982	2;3;4	3	2;4
9983	2;3;4;6;7	6	2;3;4;6;7
9984	4	4	
9985	1;2	2	1

Another class of common error is to find that Unaided mentions are not consistent with Aided. For example, if the code frame for both Aided and Unaided is 1 to 5, and a case has data as

Unaided = 3,4,5
Aided = 1,2,3

then if literally interpreted and accepted the respondent has forgotten about brands 4 and 5, which were recalled unaided just several questions prior. This is not possible for a *compos mentis* human. Aided awareness questions should always be a superset of Unaided for codes in common – otherwise recall has out-performed recognition. The check sum for this condition is Aided - Unaided >= 0 for each common brand for each case. If anywhere less than zero, then the aided/unaided data for that case is not sensible.

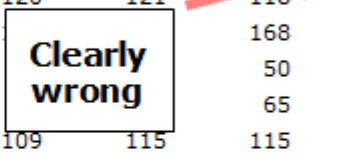
3. ZeroSumTest

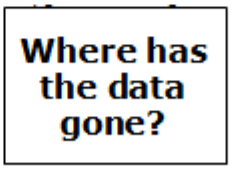
At each update, run a table of N recent Periods by the total of each variable, and examine each row (one per variable, so often there can be a few thousand rows) as frequencies (never as percents), looking for out-of-character behaviour such as a big jump in either direction, slabs of empties where data was expected, data which was not expected where variables have been seasonally rotated out, etc.

This test is especially useful for catching a failure to merge coded verbatims back into the main data set.

Top: Week
Side: ZeroSumCheck

Frequencies	Week				
	18Oct2004	25Oct2004	1Nov2004	8Nov2004	15Nov2004
sum#Where Usually Buy (Any)	50	50	50	50	50
sum#Where Usually Buy Supermarket (Any)	23	26	23	16	315
sum#Top of Mind Brand Awareness (Any)	50	50	50	50	6550
sum#Unaided Other Brand Awareness (Any)	126	120	121	118	123
sum#Unaided Brand Awareness (Any)	176			168	170
sum#Net Top of Mind Brand Awareness (Any)	50			50	50
sum#Net Unaided Other Brand Awareness (Any)	68			65	57
sum#Net Unaided Brand Awareness (Any)	118	109	115	115	107
sum#Top of Mind Ad Awareness (Any)	10	9	13	11	14
sum#Unaided Other Ad Awareness (Any)	15	14	22	18	21
sum#Unaided Ad Awareness (Any)	25	23	35	29	35
sum#Net Top of Mind Ad Awareness (Any)			13	11	14
sum#Net Unaided Other Ad Awareness (Any)			15	11	14
sum#Net Unaided Ad Awareness (Any)			28	22	28
sum#Unaided Awareness (Any)			194	184	182
sum#Net Unaided Awareness (Any)			118	118	112
sum#Aided Brand Awareness (Any)	73	75	71	73	
sum#Aided Ad Awareness TV (Any)	51	56	46	56	
sum#Aided Ad Awareness Other Media (Any)	8	19	17	15	





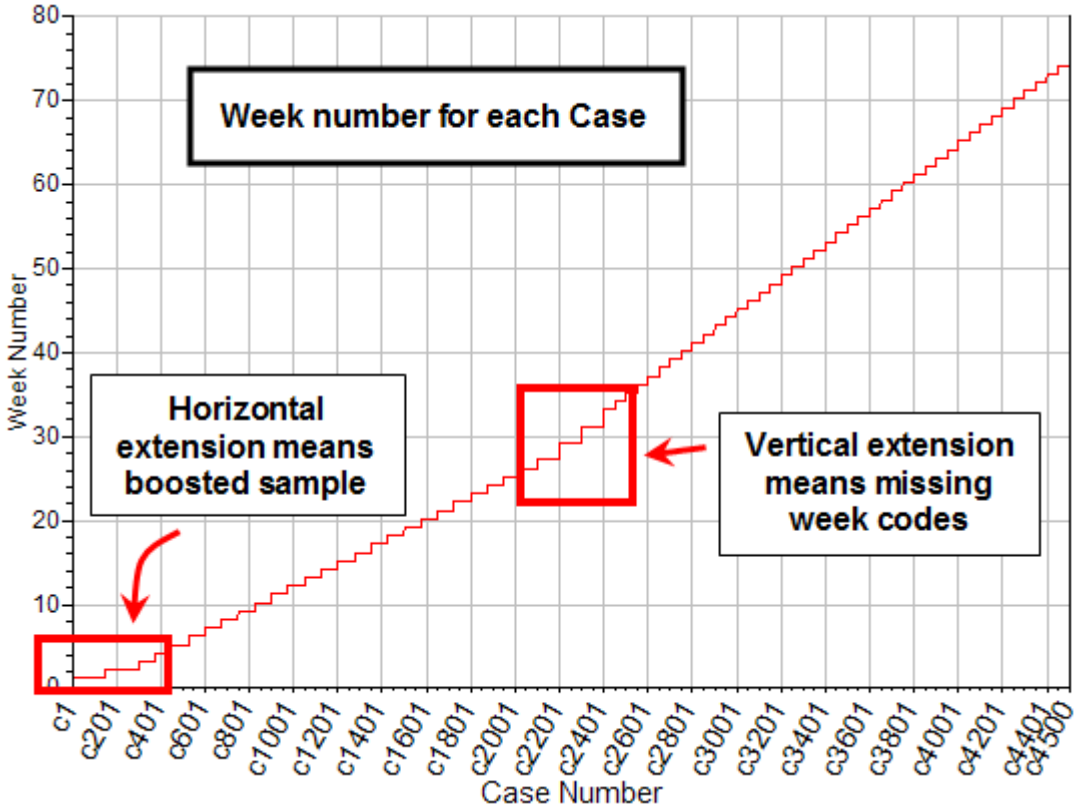
4. Compare an Update to Average of Prior N Updates

This is the detailed version of the Zero Sum test above. At each update, run a table of N recent periods by all variables, perform tests such as standard deviation of the last N periods (a large value means a lot of jerkiness), ratio or difference of this period against the average of the last N periods, etc, and then sort on the test vector. This will push all aberrants to the top and bottom row positions in the output, making it easy to skip over the boring middle and focus on the big movers in either direction.

5. Create charts which announce data errors

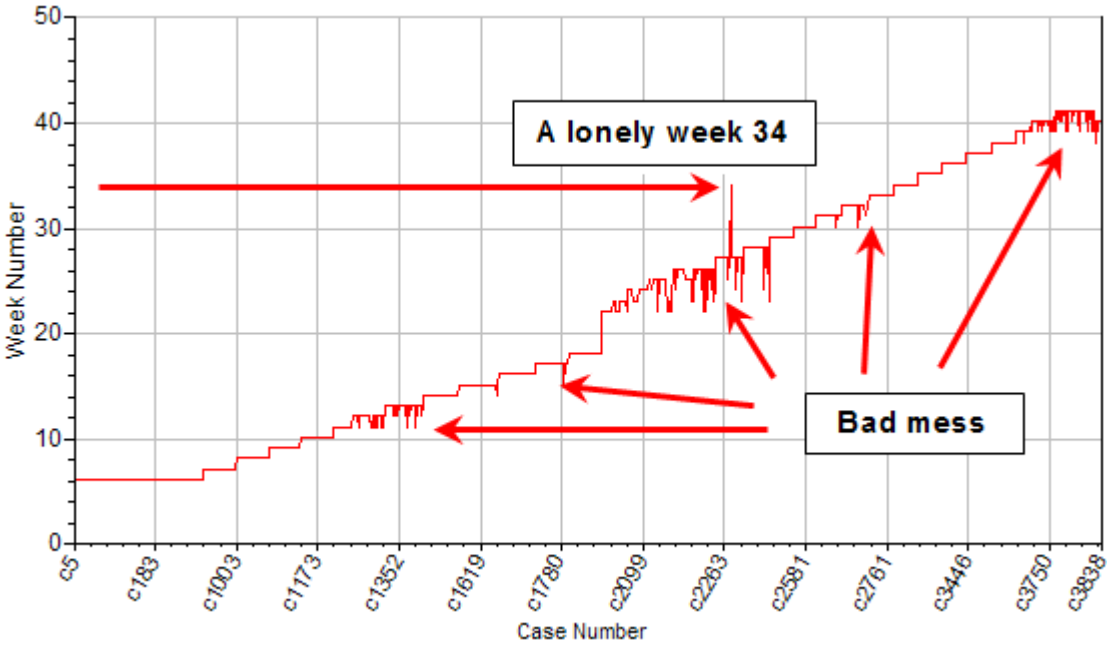
Depending on the job, there may be more candidates for this test, but two which will always apply are a chart of respondent counts within each coded period, which should show that the data is in chronological order, and a histogram of weights, which should show that the weighting regime is sensible, with no cases allocated huge and distorting weights.

For checking the period codes and the respondent counts in each period:



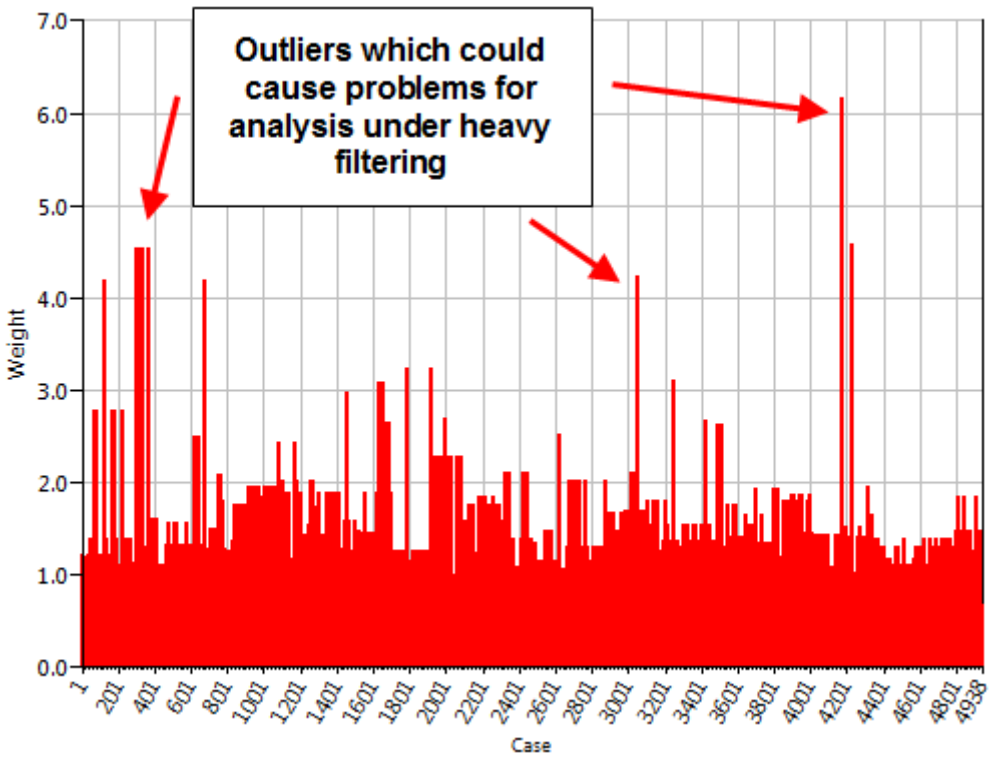
All week codes are in increasing order, but some week codes have been skipped.

An example of a very bad sequence of week codes is



The messy bits could be hiding a multitude of data sins, and should never have been allowed to happen.

For weightings:



6. Variable Reports

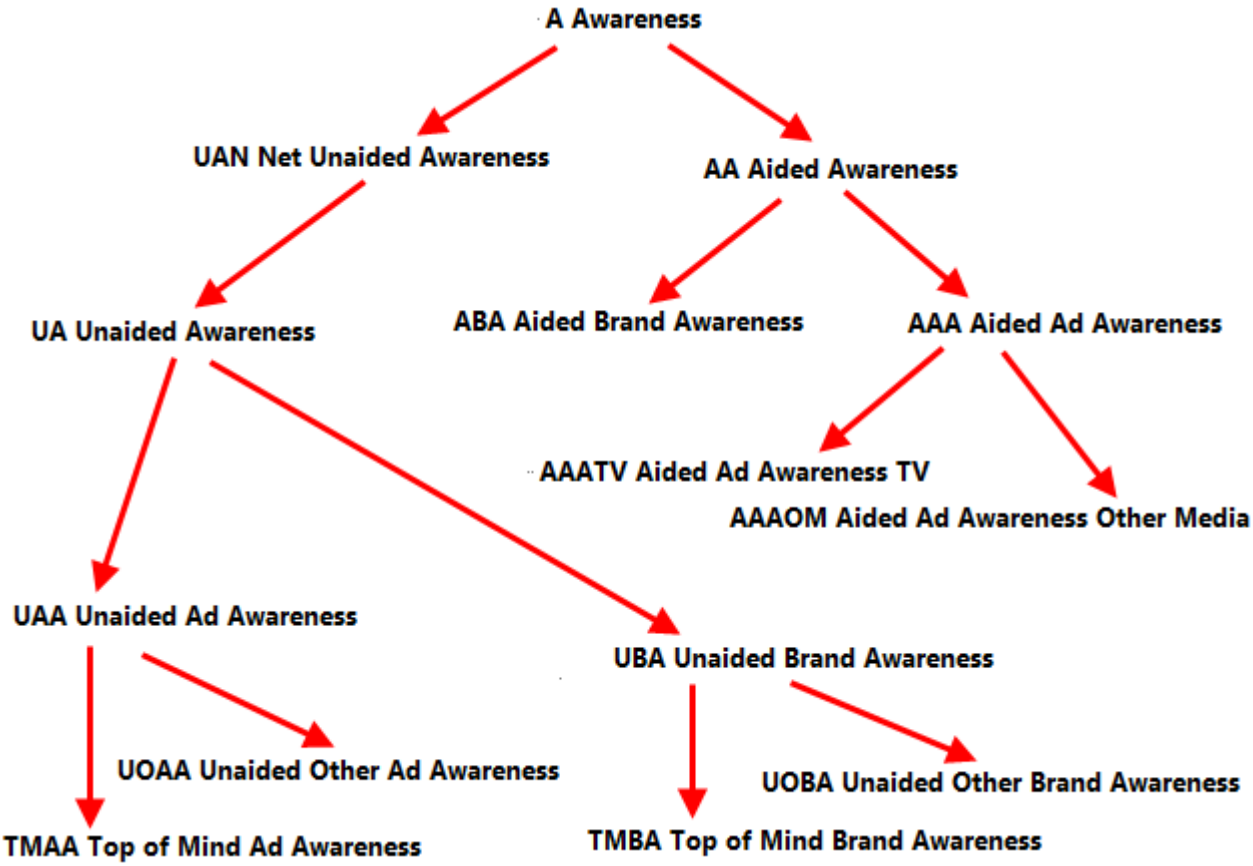
Run a process to deliver a summary report on all variables for things like number of codes, number of cases, last update, most recent data, etc. This is an important aspect of housekeeping, and a good way to purge the job of unwanted constructions, bad experiments, unused rim weights, etc.

Heteromorphism in Practice

Continuing on from the section above on software and system requirements, the theory is all very nice, but how exactly is heteromorphism achieved? The basic requirements are reference lists, self-modifying constructions, report axes which automatically collect and display new codes, a solid scripting environment for laying out and executing all routine processes, and a self-updating reporting regime.

Constructing

Consider the following derivation of a measure for Awareness from all sources: unaided, aided, brand and advertsing.



The final measure, A=Awareness, depends on 13 prior variables: seven source variables at the leaf nodes and six intermediate constructions. In an ideal system a new brand code would be seamlessly accommodated through all the steps, and then automatically appear on the relevant reports.

To automate the derivation of A against a new brand, the leaf variables must use a master reference list to collect all new codes, and all non-leaf variables must be created and maintained by specification generators for variable nets (eg UBA = TMBA or UOBA) and code lists for variant to parent nets (here UAN from UA).

Lock Report Axes

A report which plots all the brands of a measure should automatically pick up any new brands which occur. Similarly, a time series report should pick up all new period codes, and optionally drop off old ones. A brand comparison report, however, should most often stay as specified. Some reports therefore need one or both axes to be heteromorphic. In a report of week by key measures, week will be heteromorphic, but the key measures, having been carefully chosen, will not. A report of Top of Mind Awareness by Brand Last Bought will usually be heteromorphic in both axes.

Scripting

Those who began life as data or market analysts a decade or more ago, and who had to learn those arcane languages for SPSS or SAS, or the idiosyncratic Surveycraft or Quantum specification systems, are unlucky to have missed the much cleaner approaches afforded by modern scripting languages. The Windows operating system comes with scripting in either VBS or JavaScript built in. This facility has been thoroughly

exploited by SPSS Dimensions, and I thoroughly endorse the Dimensions approach. A similar line has been taken by Blaise, among others. Any modern MR software which does not support modern scripting is doing its users a profound disservice. Scripting imparts huge control to the script writer. Batch processing can be done either table by table, or can be generalised using self-filling arrays and loops etc. Scripting is where DP get to be imaginative and try new techniques to improve throughput or diagnostics. Running CT jobs should never be boring, because each one is a working laboratory on how to streamline ever more efficiently.

Any task which has to be done routinely at each update should be considered as a candidate for scripting – even such simple things as moving files around to source directories for reading, and then to backup directories post update. No matter how trivial the task, if it can be scripted, then script it.

Reporting

Finally, one of the last great manual tasks is updating the reporting instrument, most commonly a PowerPoint slideshow. Typically, the PowerPoint slideshow accumulates more annotations, comments, callouts, and graphics over time, with possibly hundreds of reports embedded in slides. How are all of these charts and tables going to be updated? There are quite a few systems which will simply populate a PowerPoint deck with tables and charts (this is a 'push' operation), but if those tables and charts then have to be manually copied to their positions in the master deck then little is gained.

The ideal system requires a way to make the PowerPoint deck update itself simply by pointing it at a set of updated reports. This is a 'pull' operation.

It is important that the difference between push and pull is very clearly understood. A push will not save much time, and will require a lot of manual effort still. A pull, on the other hand, is a fully automated process.

Conclusion

In many ways, CT has come a long way in the last twenty years, but far too often the potentials afforded by advances in software and processing systems are not being properly utilised. I am sure there are pockets of excellence throughout the industry, but too often the brute force approach reminiscent of the early years still holds sway, and the large MRs are no exception to this. The answer, as I have tried to make completely clear, is a combination of rigorously enforced work practices, and software which is adequate to the task.

Once the basic plumbing is in place, analysts can be freed to work much more as social scientists than as account trouble-shooters, and traditional DP can move beyond fixing field problems and tedious batch table runs and into architecturing the processes and procedures for job updates. This will create a productivity dividend which can be taken as either deeper analysis, or more clients.

For the future, there remains a lot of potential in looking at automating the coding of verbatims, and as desktop PC capacity increases, there will be a lot more scope for automated data mining techniques.