NLP For Testers
- The Meta Model

Alan Richardson

Compendium Developments

www.compendiumdev.co.uk/nlp

This paper will provide a brief introduction to:

- The 'Meta Model', a model of psychotherapeutic communication
- The 12 communication violations identified by the 'Meta Model'
- How testers can use the Meta Model

'If we spend our time looking for causes instead of structure we may as well give up the idea of therapy and join the group of worrying grandmothers who attack their prey with such pointless questions as "Why did you catch that cold?" "Why have you been so naughty?"'
Fritz Perls,
The Gestalt Approach

What can the NLP Meta Model do for testers?

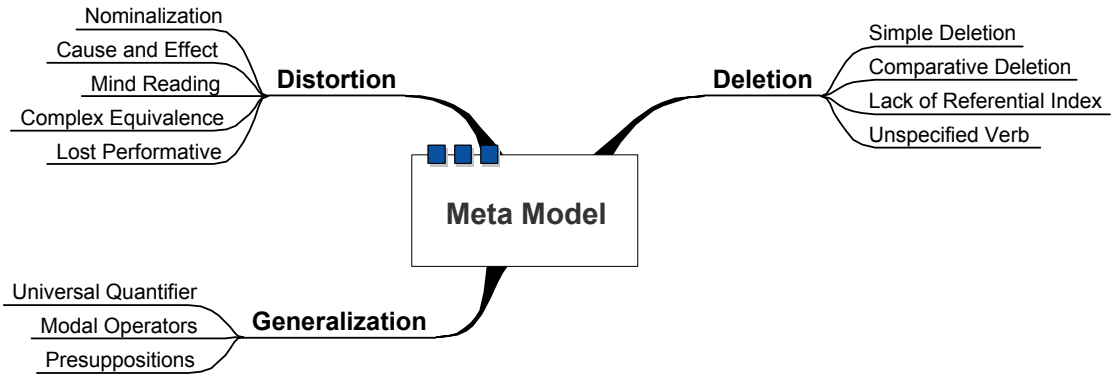
> *"...the goal of the Meta Model is not to find the 'right' answers, but rather to ask better questions – to widen our map of the world rather than to find the 'right map of the world'. The purpose of the Meta Model inquiry system is to help identify missing links, unconscious assumptions and reference experiences that make up the 'deeper structure' of our conscious models of the world."*
>
> NLP Encyclopaedia,
> Robert Dilts

The process of software development is a process of human communication. Beliefs about software are communicated as requirements. Software specifications are based on the requirements and are in turn communicated to programmers. Programmers communicate their beliefs about the software specifications to the computer in the form of program code.

Each of these communication steps may involve several different people and different communication media, both written and verbal. Each communication step is subject to ambiguity, and ambiguity can result in software defects.

The Meta Model provides testers with a simple model of:

- 3 ambiguity generating transformations – deletion, distortion and generalization
- 12 easily identifiable communication violations resulting from the transformations



The paper will provide an overview of the NLP Meta Model and explore ways in which knowledge of the Meta Model can be useful to testers:

- Apply the Meta Model to the communication we receive, identify ambiguity and possible mistakes early, before being encoded in the system
- Apply the Meta Model to the communication we give to,
  - Improve defect reporting
  - Help people understand what we aim to do
  - Improve development team/test team relationships
- Apply the Meta Model to the system to identify areas of test and as a test derivation aid
  - "The docs say the system will *always* do this, if I can find a situation under which it doesn't then..."
  - What is presupposed by this dialog? How can I invalidate that presupposition?
- Teach testers an easy, effective and repeatable approach to documentation analysis
- Apply the Meta Model to our beliefs about our environment and provide a framework for thinking about our context of testing

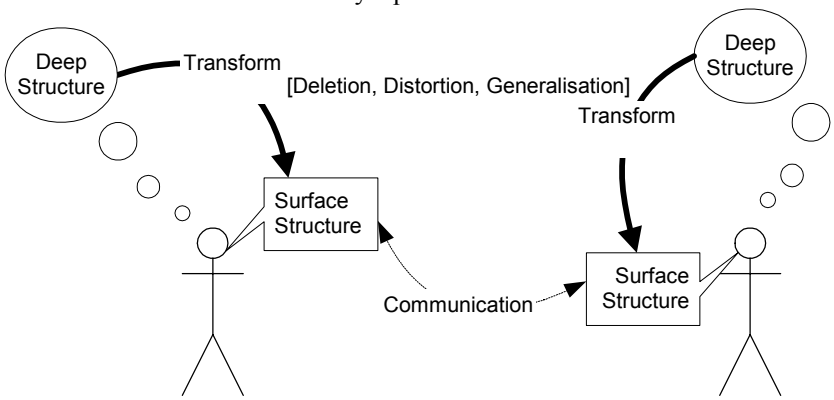
What is the NLP Meta Model?

The NLP Meta Model is a system of inquiry to explore areas of communication that are susceptible to ambiguity.

An introduction

"The Structure of Magic" [SoM1], by Richard Bandler and John Grinder, is a study of the language and the application of language in therapy of the 1970s. [SoM1] provides a formal model of the language used, enabling it to be easily understood, taught and used by therapist and non-therapists alike.

The formal model produced, called the Meta Model, was constructed using the theories of transformational grammar. Under transformational grammar, our communication, what we say and write, are surface structure representations of a richer model of the world (deep structure). And we map our deep structure onto the surface structure by a process of transformation.



NLP Deep/Surface Structure Model

For example, if I tell you that "I bought a computer". Then the statement "I bought a Computer" is a surface structure representation of a richer deep structure. The deep structure contains all my memories of buying a computer, including:

- The strategies I used to decide which computer to buy,
- Memories of all the computers I discarded before choosing the one that I chose,
- How I felt buying the computer,
- How much it cost,
- If I thought it value for money,
- What the shop/salesman was like,
- Where I bought it from,
- What the shop smelt like,
- How I felt carrying the computer home.

All of this information has been deleted in my transformation from the Deep Structure to the surface structure statement. But all that information is accessible to you if you *ask the right questions*.

Over time, my computer may crash, have problems with the hardware, and I may come to feel that the computer wasn't as good a deal as I thought and I may start saying "I bought a bad computer" *and* may start feeling bad about it. At this point I am not only deleting information, about it being the best computer I could buy at the time, I am distorting my deep structure; and that distortion is having an effect on my real world interaction. But the deep structure is still accessible, and with the correct questioning, I may come to realise that "I bought the best computer at the time", and that "the computer I bought, now has some problems" which is a different surface structure representation of the same deep structure and may have a different effect on my outlook.

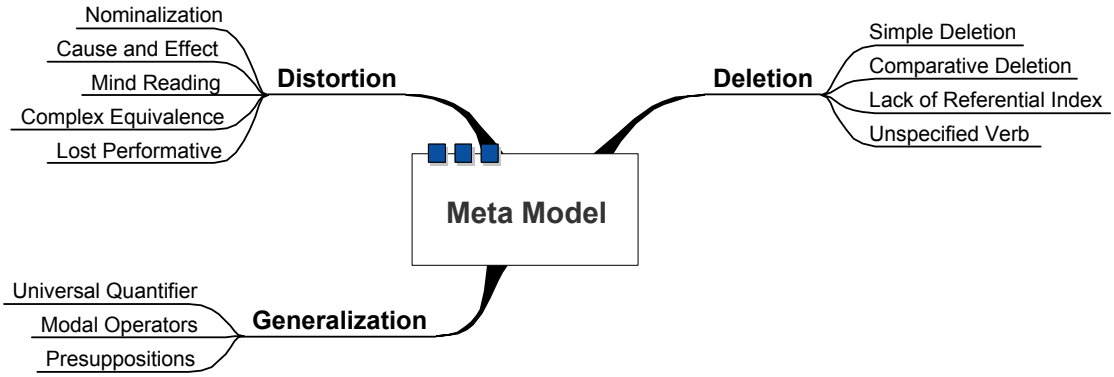
The Meta Model categorises the transformations from deep structure to surface structure as deletions, generalizations and distortions.

Recognition of Distortion, Deletion and Generalization in a communication allows us to apply a high level analysis to a statement and ask questions to identify:

- What is missing from this communication that might cause problems,
- What is distorted in this communication, and presented, without adequate definition
- What is generalized in this communication and what is implied by that generalization

The Meta Model categories are further divided into 12 elements: nominalization, cause and effect, mind reading, complex equivalence, lost performative, universal quantifier, modal operator, presupposition, simple deletion, comparative deletion, referential index and unspecified verbs. Each of these elements will be discussed and explained later in the paper.

Each of the 12 elements correspond to recognizable language patterns that are found in human communication and have associated questions that can be asked in order to derive more information about the deep structure associated with the communication.



Categorisation taken from the Appendix of 'Persuasion Engineering' [PE95]

Some reasons why all of this is important are:

- Human communication can be ambiguous
- Ambiguity can adversely affect the communication and the communicator
- It is possible to be taught to recognize ambiguity
- It is possible to respond to ambiguity to gain clarification
- Clarification can help both the communicator and the communication
- Software development is a process of human communication

Most software testers will have encountered requirements, which seemed to be understood by everyone but, which every role in the development process had a different interpretation of. This probably resulted in software defects, delays, unhappy users, rework and a whole host of follow on problems.

This is not just a problem with requirements; specifications, defect reports, test strategies, in fact all communication in the software development process can be impacted by ambiguity.

Communication problems in software development are not limited to the artifacts produced and the 'Chinese whispers' effect along the software development lifecycle. Many inter-team problems are a result of communication issues, and a practical knowledge of the Meta Model can help testers understand and deal with these issues.

This paper primarily concentrates on artifact analysis with the Meta Model because:

- I am not about to train you to be a therapist, because I am not a therapist.
- The use of the Meta Model to deal with human belief systems is well covered in the NLP literature.
- The application to artifacts and software systems is a novel use of the Meta Model, and easily applied by Software Testers.

More information on the Meta Model and Software Testing

Now that I have provided an overview of the Meta Model, I will describe some correspondences with Software Testing.

In much the same way that testing is an open-ended task, there is always another test that could be run. The Meta Model is open-ended in that it is a method of inquiry and each application of the Meta Model elicits another surface structure description of the client's deep structure. This surface structure description can then be subjected to another Meta Model inquiry.

Just as we do not test for the sake of testing, the Meta Model is not use to get information for the sake of information. Testers get information to effect change in the development team's beliefs about the system. The Meta Model is used to get information to allow therapists to effect a change in the client. Testers can ask questions to get information if it seems to them that a transformation has resulted in a communication that may be ambiguous enough to cause problems of understanding.

When I analyse previous testing projects that I have been on from the point of view of the Meta Model, I find that much of the testing use I have derived from the Meta Model is associated with Generalizations and Deletions. I have designed tests specifically to target these two types of transformations.

Many of the requirements and specification issues that I identify have also associated with Generalization and Deletion, and I have asked simple Meta Model questions to have them clarified. This has resulted in the nature of the testing about to be performed changing and helped achieve better collaboration with other roles involved in the development process.

However, much of the inter-team communication problems have been recognised by distortion.

> **Note**: Each of the Meta Model elements are not *wrong* or bad. Depending on the context, they may be highly appropriate and exactly *right*. They *can* be sources of ambiguity, and ambiguity, in software development communication, *can* be a source of error.

Generalization

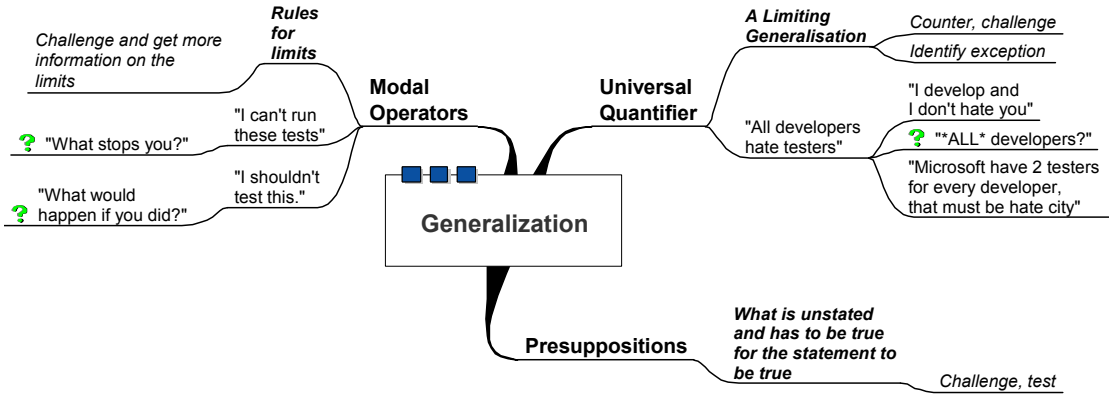
Exceptio probat regulam in casibus non exceptis
Cree's Dictionary of Latin Quotations,
Anthony Cree, (Newbury, 1978)

Generalization is a process of building new rules or models from parts of an experience.

I have encountered the results of unidentified processes of generalization in software construction as:

- Special cases being introduced as code changes during the testing phases, as the testing process identified more and more exceptions. This quickly led to chunks of code that were more complicated to test, prone to failure, and difficult to maintain.
    - In this case had the generalization been tackled early then a different implementation may have been provided which would probably have been less error prone.
- Situations that the system simply could not handle because the architecture was built around presuppositions that were simply untrue.
    - For example a file based system that failed regularly due to file corruption and contention because of multiple processes accessing the same files, a database solution would have been a better solution had the presuppositions been identified, questioned and acted upon.
- Situations that caused the system to crash because the limiting rules, around which the system was designed, were not enforced.
    - Incorrectly formatted files being fed in to the system causing it to crash

I have found generalizations to be one of the most useful areas of the Meta Model to apply to Software Development communication. We will all have had experiences where flawed generalizations were unchallenged and carried through to the software itself, and resulted in problems. This section will explore how to identify and question generalization.



Presuppositions

"...Up until this stage, a grandmaster's thoughts have been based on general ideas and strategic principles. Now, at long last, he will start looking for the best next move. He will establish what moves are possible, and how they fit in with his plan. Then he will begin analysing many variations. For each of the moves he will examine he will foresee the opponent's reply, then his best answer and so on. Only after finishing this immense task, now purely analytical, will Smyslov move a piece and stop his clock. Thinking over, move made!"

Think Like a Grandmaster,
Alexander Kotov, 1971

A presupposition is an element in a statement that must be true for the statement to be true.

Presupposition Analysis is one way to develop the skill of looking at a system, working out how it could go wrong, and then identifying '*THE* test' to expose the problem

When I receive a communication from someone. What is presupposed by that person's statement that has to be true for their statement to be true? Because if any of the presuppositions are not true, then that may invalidate that person's overall statement, and if the statement is one which is not helpful to their world view then we can go some way towards helping them explore a new world view, give them choices and conduct effective therapy.

But this paper deals with Software Development so… What are the presuppositions in the following requirement?

> "The daily trade file must be processed in under 5 seconds when the system starts up"

I have listed some of the presuppositions below, with some of the responses that a tester might have to those presuppositions:

There is a daily trade file	What if it doesn't exist?
The trade file is generated in some way	By what way? Which system?
The trade file is accessible	Is it on our environment, where does it reside? What if the permissions are set wrong?
The trade file can be processed	What if it is corrupt?
The trade file is a file	What kind of file? Binary, Text?)
The system will start-up on a daily basis	What if the system doesn't start up one day? Does the trade file get overwritten or something worse? What if the system starts up more than once?

A fuller analysis of this requirement statement appears in the Appendix 'An Annotated Meta Model Example'.

The above requirement is an amalgam of a number of different requirements from past projects. The ambiguity in similar requirements caused problems in the system implementation because some of the questions identified were not asked or answered on those projects.

- Presuppositions may be presented as assumptions that are so taken for granted that they don't need to be checked. Often they do need to be checked.
- In requirement documents you can often identify unstated requirements by examining the presuppositions. If the unstated requirements are not picked up early we can be involved in a project where "the users' don't know what they want" and "the requirements keep changing".
- Presupposition analysis can help identify potential problems early as the analysis results in a deeper exploration of the problem domain and Presuppositions can be questioned before the application is built.
- Presuppositions can be the target of challenges in the form of tests.
- Presupposition analysis can help us identify preconditions for tests, and help us to order the tests. What is presupposed by a test? Check those presuppositions first; we may have not identified a fundamental test.

Presuppositions are familiar to us when we go through the process of empathising with someone and trying to figure out why they do the things they do. "What would I have to believe in order to act like that?"

Finding *THE* Test

If I am faced with a screen, or a requirement, or a set of functionality in a system, and I start thinking about the presuppositions associated with that. An approach to testing is to see if I can invalidate any of those presuppositions.

Miracles in therapy occur when the therapist asks a single question of the client that leads to an emotional epiphany.

'*THE* Test', is the Software Testing equivalent. What is the one test you can do to the system that will make plain a fundamental error of the implementation? In one sense Tests are just questions that we ask the system.

We can generate 2nd, 3rd, 4th, …, order presuppositions by subjecting our presuppositions, and the questions we ask of those presuppositions, to presupposition analysis. The quote from Kotov [AK71] at the head of this section illustrates the process of bifurcating presupposition analysis well.

By working along a chain of presuppositions and questions we arise at a presupposition at a more fundamental level, and if that presupposition fails our question then we are more likely to find a test that breaks the system. This is one way that experienced exploratory testers are able to sit down at a system and break it very quickly and very visibly.

Example:

"The daily trade file must be processed in under 5 seconds when the system starts up"

- 1st order presupposition: "The daily trade file must exist",
- Question: "What does the file contain?" Answer: "information that can be processed"
- 2nd order presupposition "the contents are valid"
- Question: "What happens if I invalidate the contents?" – *I'll test that*

An example of how to use this presupposition for testing can be found in the testing literature: James Whittaker's 'How to Break Software' [JW03] in 'Chapter 4: Testing from the File System Interface'. 'Attack 6: Vary or corrupt file contents'.

The notion of presuppositions can be used on GUI systems too.

When faced with a screen, what has to be true for this screen to be true?

- If the screen is a list of items then
  - The item displayed exists
    - What if it doesn't?
      - Can we create a state in the system where we display the list, but delete an underlying record displayed in the list, then try and open that record?
  - The item displayed must have those details
    - What if it doesn't? What if it has been changed elsewhere?
      - Can we create a state in the system where we change the underlying record details, then try and open the record in the list? Are the underlying details displayed, or are the details in the list displayed?

Testers may learn to do this as they become experienced with testing. Sometimes experienced testers have the appearance of following this process, but apply the test because they have encountered the defect before, rather than because they are thinking through the presuppositions. For me, learning 'presupposition analysis' has allowed me to approach exploratory testing more consciously, practice it, use it more often, and get more benefit from it.

Universal Quantifier

(All, Never, Every) M : 1 (Just that, Only)

"They ***never*** listen to me", "He ***always*** puts me down". In a therapeutic context, these statements can lead to the client having limited beliefs about their world, which the therapist can challenge and expand the range of possibilities open to the client.

When universally quantified statements are present in software documentation "The system will always…" "The system will never…" then this leads to a number of obvious thoughts that the tester can consider:

- These kinds of statements are crying out for a *counter example*. The statement may be a limiting belief about the software that should be challenged early in the project, to ensure that the exceptions have been considered.

- Specifically constructing tests to try and identify any exceptions to the statement, exceptions that would then be classed as defects.
- The tester may need to consider a test data strategy based on combinatorial test data generation techniques (orthogonal arrays, allpairs) to provide comprehensive data sets to test against.
- Boundary Value Analysis, or more specifically, Domain Based Test techniques can be used in this situation. We may well be dealing with an infinite domain, or we may be dealing with a domain that hasn't specified the boundaries.

Part of Test Process Improvement involves examining our beliefs about the Software Test process and identifying which beliefs are useful and which are not. I have seen situations where a test process has reached a high level of *maturity* with rigorously defined processes and strict rules about how testing *must* be conducted.

Effective TPI deals with appropriateness. And it is useful to examine the Universally Quantified rules in a test process to see how universal they actually are.

- "Never let software testers see the source code"
- "Never let the same person that coded it, test it"
- "Always write a test script before running a test"
- "All tests must be cross referenced to requirements"

We may discover that we cannot readily justify these statements on all our projects and that our process of TPI has gone a step to far and generated generalizations rather than increased our ability to do more effective testing.

Apply the Meta Model to the statements that are documented in your test process or strategy, and to the statements you make that represent your beliefs about software testing. Do any of those statements suffer from a generalization transformation?

Modal Operators

Should, Must, Could, Need, Might

When dealing with people, modal operators can be used as motivators for change. We can achieve different results in communication by changing the modal operator in a statement: "I *must* wash the dishes", "I *should* wash the dishes", "I *could* wash the dishes", "I *need* to wash the dishes." Try it out for yourself, some statements motivate you to do it, others have presuppositions of choice and you might choose to ignore it and put it off, others might build feelings of resistance due to the language used.

In requirement documents there are modal operators of possibility that can add ambiguity. {Might, May, Often} "The system might need to reset itself" Under what circumstances? Does it Always do it under those circumstances or just sometimes? Is it random?

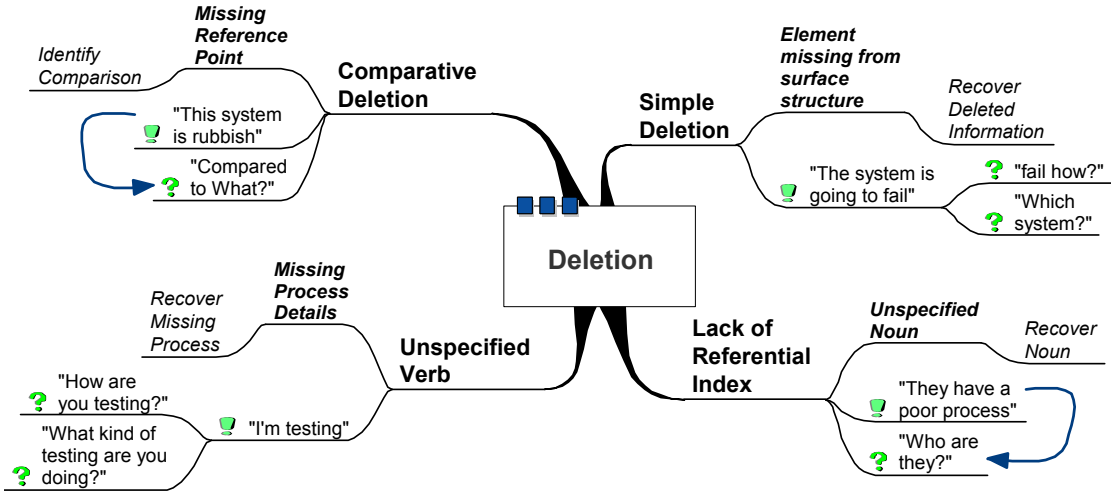
Being aware of the modal operator when reviewing documentation allows us to ask specific questions to quantify requirements when necessary to remove ambiguity.

Note: Use of the Meta Model does not have to be done in a way that bludgeons the people you work with an endless series of questions.

- Skilled Testers have to be able to work with some level of ambiguity and missing information.
- Skilled testers analyse what they know and learn to identify if the current level of ambiguity in their model is acceptable, and then assess and report on the level of risk posed by that ambiguity.
- Sometimes questions may not be answered, and testing may have to continue anyway.

Fortunately, testers can pose their questions as tests and ask the system.

Deletions



At the end of an information gathering interview or meeting I often ask a question similar to: "Is there anything that you not told me, that my not knowing, will prevent me from being able to do what I need to do to help you?"

I have long forgotten the original source where I learned of this question, and subsequently cannot attribute it for your future study. But I have found it enormously useful when consulting, and the question relies on the identification of deletions for its impact. By asking this question I am searching for deletions outside of the scope of what I was able to identify from the responses in an interview.

Have you ever encountered a situation where you didn't test an area of the system because you didn't know anything about it? Deletions can cause major problems in the software process.

Simple Deletion

A Simple Deletion is information that is missing from a statement. Information that would complete the statement and remove ambiguity.

"I'm tired". We could make assumptions about what the speaker is tired of; they may have previously said they were tired about their job and we could assume that that is what they mean. They may only be tired of a specific aspect of their work and in fact may love their job. So we target the deletion to get more information. *"Tired of what?"* "Well, I did a lot of DIY yesterday, I'm glad to be back at work so I don't have to paint any more."

"The input file is received via email" has an obvious deletion, "received from whom?"

There are slightly more subtle deletions in the statement "Sent to whom?", "received when?" and there are a whole set of deletions about the input file:

- What format is the file?
- What size is the file?
- What does the file contain?

The hardest deletions to spot, and the ones that give most problems, are those that have no indication of their existence in the surface structure statements. That is why the question at the start of this section "Is there anything you have not told me…" is so powerful.

We can sometimes spot these deletions by building a model of the system that we build from all the questions that we ask . And then applying the Meta Model questions to our model. It may be that only by putting together all the surface structure statements that we can identify a deletion that will cause a

more fundamental problem. Some deletions may be identified through a process of presupposition analysis.

Comparative Deletion

"You're Finding too many bugs"	}	Compared to Who?
"You're not finding enough bugs"		Compared to What?

A Comparative Deletion occurs when the reference source is not defined. When a statement containing a comparative deletion is communicated, the statement may be received as a value judgement instead of a comparison and can be a source of dispute.

Sometimes things are held up to comparison in such a way that only the negatives are noticed and that none of the superior attributes are acknowledged. This can be seen on development projects when comparisons are made to estimates given at the start of the project and then used to bludgeon staff with later in the project.

- "You haven't run enough test cases."
- *Compared to what?*
- "***This*** estimate",
- *Ah, the estimate we produced before knew what we were doing and before we new the real scope of the task. Hmm, it may be time to revisit that estimate.*

"The system is running too slowly." Compared to what? Possibly compared to what the user needs. The deletion may reveal a previously unidentified requirement.

Lack of Referential Index

A Referential Index is an object referred to in a statement. The lack of a referential index occurs when the object is replaced by: It, They, Them, We, Us, That.

The question in response to a missing referential index is to identify the object that is referred to.

"The Users will want to do this." Deletes information about *which users specifically* will want to do *this*. The notion of Roles and different types of users can be very important during testing to help identify risk associated with activities and generate test scenarios.

Unspecified Verb

An unspecified verb refers to a statement with a verb that has limited information about how the verb is carried out. "The system will process this", process it how?

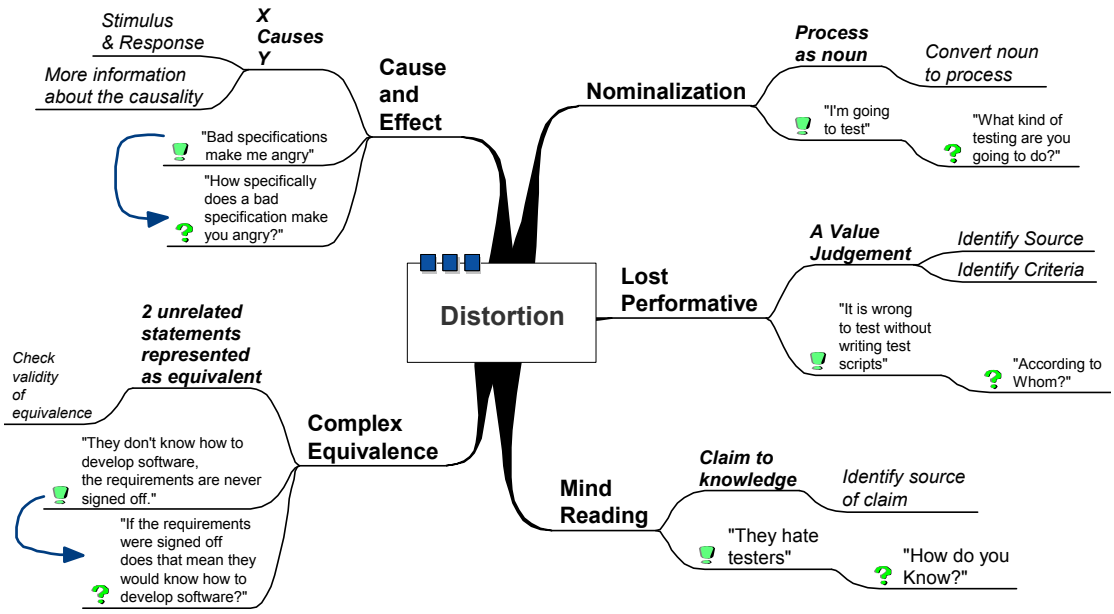
Lack of information about verbs can cause problems when testing, if assumptions are made about the deleted information.

Distortion

Distortions are a common source of assumptions.

Have you ever worked on a system where some functions were added because the software development team used Mind Reading to generate requirements? Because the developers *knew* what the users needed.

Distortion can be a source of inter-team conflict and as a manager it can be useful to identify distortions in the communication of your staff when they talk about teams that they are having problems with.



Nominalization

"Testers don't get any respect"

Nominalization refers to converting a verb into a noun.

Respect is a process. We may talk about having respect for someone, but that doesn't really mean anything. We may demonstrate that we respect someone by asking him or her to comment on our strategy. How do you demonstrate that you respect someone?

Software Development is full of nominalizations, in fact 'Software Development' is a nominalization, and we distort all the activities and roles involved in the process of developing software when we nominalize it.

A Nominalization can be an assumption on the part of the communicator that the other person has the same view of the process that they do, it may also be a cover up for a lack of knowledge about that process.

A nominalization is challenged by converting the noun into a verb and asking for more information "How do you know when someone does, or does not, engage in the process of respecting you?"

Lost Performative

"This system sucks"

Who says? According to Whom? Where did you hear that?

A lost performative refers to the missing authority from a statement. Who is it that is saying this statement, or who originally said it..

When dealing with people, we may want to know if this is a self-generated part of their model or an outside influence. If it is an outside influence then it could be countered with a conflicting statement of someone equally authoritative; but maybe neither authority's statement is true. If it is an internally influenced statement then we can trace the chain of belief and offer an alternative view (if this structure is not helpful).

A Lost performative may lead on to Mind Reading:

- "The system really must do this"
- *Who says?*
- "The users do"
- *Where did they say this?*
- "Look I just know OK."

Mind Reading

"The development team don't respect us testers" } "How do you know?"

Mind reading is any claim to knowledge where the source is unknown.

In verbal communication, mind reading can lead to assumptions which create unproductive beliefs about the situation, when the situation may not really be like the communication.

In written documentation, particularly requirements documentation, mind reading can lead to assumptions of authority. One justification for test processes insisting on cross-referencing tests back to requirements is to demonstrate an authoritative source for those tests.

Unsubstantiated statements might mean that the requirements analyst has put words in to the mouth of the user and we may have extra functionality and requirements that are not strictly required and may detract the development team from producing the requirements that are necessary.

Testers should be wary about applying Mind Reading to the systems they test:

- "Great the record got saved. On to the next test."
- *How do you know the record got saved?*
- "Well, the progress bar showing it was being saved was displayed."
- *Have you looked in the database?*
- "Well, no but…"

Sometimes we *know* something has happened but we don't actually have an authoritative source for that information. When we identify Mind Reading, we question the communication to identify a source.

Cause and Effect

Cause and effect is a chaining together of statements, leading to some evidence for a generalized conclusion. An X causes Y statement.

"I saved a file and the system crashed"

A cause and effect statement may need to be questioned to get more information on exactly How X causes Y, as we question this we may identify a series of deletions in one of the follow on surface structures.

Cause and effect brings to mind the Stimulus and Response theories from behaviourism [JW66].

By thinking in behaviouristic terms we can question the conditioning associated with the stimulus and the response.

Behaviourists experimented with conditioning a subject so that the stimulus would trigger a different response. So with that in mind we can ask questions of a Cause and Effect statement to check this:

- Does X always cause Y?
  - We might try to use the process of testing to try and condition the system into generating a different response from the same stimulus.
  - What environmental factors might affect this stimulus or this response?
- Are there any situations where X doesn't cause Y?
- Is the response only triggered by that stimulus or are there other ways to achieve that response?

Behaviourists also experimented with conditioning by over exposure. Is it possible to continually trigger stimulus X until the response does not happen? In computer systems this may be the result of a memory leak for example, or overload of the system in a denial of service attack.

Complex Equivalence

"The development team don't respect us testers, they never give us signed off design documents early enough to write our test scripts."

Complex Equivalence is a causal model that results in a conclusion. The statements that make up a complex equivalence are often not joined with a specific conjunction (and, but, because), but are presented in sequence with an implied conjunction. E.g. "They don't know what they are doing, the requirements are never signed off." We can suspect that there may be an equivalence between those two statements in the speaker's model. And we can question it by testing the equivalence. "So if the requirements were signed off, that would mean that they know what they are doing?"

A source of problems in communication can result when the equivalences aren't valid.

Testers should watch for complex equivalences in their defect reports as it is very easy to cause conflict with the development team if the equivalence conclusion is not one that is shared by developers.

"The system crashes every time I open a file, the file load routine is broken". Well, maybe it isn't, all we can say for sure is that the system has crashed each of the different ways you tried to have it open specific files. And we can document those ways to help the person acting on the report recreate and fix the problem.

A perfect way to create conflict in a software development team is to use cause and effect in your defect reports and document a generalized conclusion statement that is not shared by the development team. If you really want to annoy them try "The system crashes every time I open a file, the file load routine is broken because you are lousy developers".

Reflections on the Meta Model for Software Testing

"…our Meta-Model does not, by any means, exhaust the choices or possibilities of what a therapist might do in the therapeutic encounter. Rather, it is designed to be integrated with the techniques and methods in already established fields of psychotherapy. The integration of the explicit Meta-model with the techniques and methods of therapy in which you are already skilled will not only expand the choices you have as a therapist, but it will increase the potency of your style of therapy by making the interventions you use directed explicitly at expanding your client's model of the world."

The Structure of Magic, Volume 1, page 157

Having gone through each of the Meta Model elements I am now going to reflect on the Meta Model and software testing in general.

Software Testing As A Questioning Model

Software Testing can be viewed as asking questions of the system under test.

How does an effective therapist to know what question to ask next of the client? How does an effective Software tester know what test to carry out?

A therapist that knows the Meta Model can listen to their client's surface structure statements and use the framework provided by the Meta Model to ask a question based on the identified semantic illformedness in their client's statement. The Therapist takes all these responses and builds a model of their client's model of the world, and to this they apply their techniques and conduct therapy.

The tester does a similar thing. All testers build models of the system. Exploratory testers build a model as they use the system; their use of questions helps them learn more about the system in order to ask even more questions of it.

One of the techniques I used to use often when viewing documentation, but less so now, was to view it with a very simple questioning model: "Why? Where? When? What? How? Who?" This is a very generic model and it is often possible to ask all 6 questions against any statement in the requirements document. The effective use of these questions relied on my experience and understanding – otherwise I could spend a lot of time asking questions that don't add much to my accumulated understanding of the system.

The Meta Model is a more directed model and provides a more focussed set of questions to ask within the context of a specific statement. Based on the semantic illformedness in a given communication, we have specific questions that we can ask.

Skilled usage of the Meta Model depends on identifying when to ask one question above others - to know which semantic illformedness to follow up. Sometimes we may want to expose a generalization and reveal an ambiguity so that the communicators take the document away and reformulate the communication. Perhaps we want to get more information in order to provide a counter example?

A key skill for testers is the ability to answer, "What is going to go wrong with this system?" The best, and easiest way I know of explaining this skill, so that people can practise and develop it, is by exploring the Meta Model structure of 'presuppositions'.

Warning, Rapport Buster, Assumptions are not presuppositions

Sometimes when faced with a Meta Model element. We ask questions that are not associated with the surface structure presented, there is nothing wrong with this, the questions are suggestions, and we are pursuing information, so if we identify information we want, we should go after it. But the Meta Model is not a mechanism for building rapport between speakers and if used injudiously can lead to communication complications. As can non-Meta Model communication. But with knowledge of the

Meta Model you might have a better chance of figuring out how you got a specific response to your communication.

E.g. faced with the statement

"It is wrong to test without writing test scripts,"

A questioner might fill in a presupposed deletion ('first') and expand on the unspecified verb

"It is wrong to <perform test execution> without writing test scripts <first>."

Then tackle the modal operator 'first' with the question

"What would happen if you did?"

Now if the questioner has filled in the gaps with the correct assumptions then they have managed to jump ahead in the Meta Model questioning.

If they got the wrong assumptions, or the speaker wasn't ready to have their model of the world questioned so quickly, a statement of resistance might be forthcoming.

"Look, its just wrong, I can't believe you're even asking, I thought you knew what you were doing!"

At which point the questioner might want to drop their assumptions and target the surface structure violation presented (Mind Reading).

"Yeah, you're probably right, I just wondered where you got that rule from?"

Make Conscious What we do

"...to talk about it contemptuously as 'obvious.' This is where they make their biggest mistake." Fritz Perls

In some sense, the Meta Model is commonsense and some testers may look at the Meta Model and say "I do *all* this anyway" as they recognise that they have previously asked some of the questions documented by the Meta Model.

"This is all stuff I know." } Do you apply it consciously?
"This is all stuff I do as a tester } Could you teach me to do it?

The benefit of learning the Meta Model is that we can have a greater understanding of what prompts us to ask certain questions. We can return to the model when under pressure to remind ourselves of possible questions we can ask.

And we will undoubtedly ask more questions than those prompted by the Meta Model.

- How many of the questions we ask are prompted by what we learn directly from the system?
- How many are prompted by our model of the system?
- How many are prompted by our model of this type of software?
- How may are prompted by our model of software in general?

We will ask questions about the relationships between things, motivations for doing things, *ways* of doing things. How else, are we identifying what questions to ask of the system?

Asking these questions is a useful technique for exploring our own personal approach to testing.

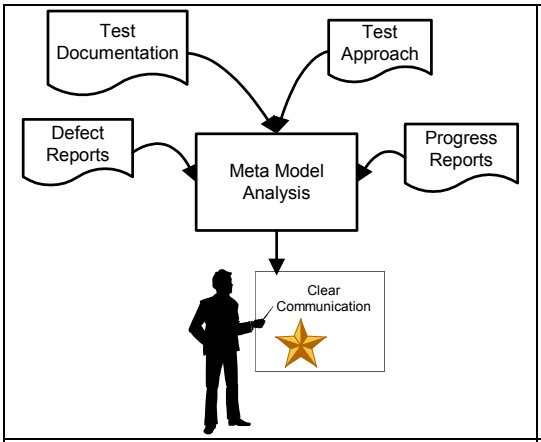
The Meta Model: by naming and describing certain violations, by giving specific questions, by giving ways of practising these questions and identifying when to ask them, allows you to apply these processes more consciously, and enables you to explain what you do.

By specifically learning the Meta Model we can gain more control over what we do as testers, to have more conscious awareness of how we choose to do certain types of tests and how we know to ask certain questions.

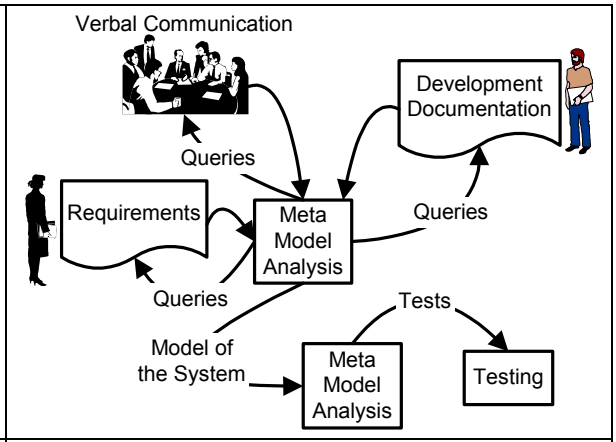
We can also more easily explain to others what prompted us to ask certain types of questions and can minimise the communication that we make to others which can cause resistance.

> **"*Exceptio probat regulam in casibus non exceptis*"** can be translated as **"The exception affirms the rule in cases not excepted"**. e.g. "Free Parking on Sundays" allows us to conclude that the rule is that "Parking is not free, except on Sundays".
>
> Generalizations are presented as statements of truth or rules. The Meta Model statements used as responses to identified generalizations are challenges to that truth:
>
> - Universal Quantifiers:. identify unstated exceptions, this may lead to the rule being changed if the exception domain is too large.
> - Presuppositions: examine the unstated parts of the rule which are not excepted
> - Modal Operators: examine the stated exceptions to the rule and determining their validity.

Communication we give and receive



Apply the Meta Model to our own work and communicate clearly



Build a clearer model, and question the system, by applying the Meta Model

We *can* gain some control over the interactions we have with others. We cannot control the people we deal with, and it may at times seem that we cannot control the systems we test. But we can control how we react to the communication we receive, we can control how we provide information to people.

The Meta Model deals with structure of communication. It deals with *how* a model of the world is constructed, not why.

Software Testing deals with *How*.

When we write a defect report and investigate the circumstances of a defect we are looking at *how* the defect occurred and what evidence we have to support it. We are not looking at *why* the defect occurred. Communicating in terms or *how* instead of *why* will change the language we use to write defect reports. If we introduce Whys into the defect report we risk alienating the reader by providing an explanation that they do not agree with. Filtering defect reports through the Meta Model can help improve the communication of those defects.

How, not Why

In the Meta Model questions there were no 'Why?' questions. Fritz Perls, and Virginia Satir have commented on 'Why?' questions thus:

Fritz Perls: *"'Why' opens up an endless series of questions which can only be answered by a first cause that is self-caused…If our patient learns the how … he can work through his interruptions into his real self and the activities he wants to carry out".*

The Gestalt Approach, page 54

Virginia Satir: *"People can ask 'how' questions instead of 'why' questions. Generally speaking, 'how' questions lead to information and understanding, and 'whys' imply blame and so produce defensiveness. Anything contributing to defensiveness contributes to low pot and leads to potentially unsatisfying outcomes."*

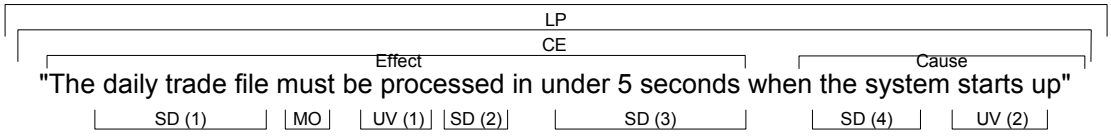
The New PeopleMaking, page 136

Fritz Perls: *"There is, however, one way of asking questions – used by most orthodox therapists – which seems to me of little therapeutic value. These are the questions starting with 'Why?'…The 'why' questions produce only pat answers, defensiveness, rationalizations, excuses, and the delusion that an event can be explained by a single cause. The why does not discriminate purpose, origin, or background. Under the mask of inquiry it has contributed perhaps more to human confusion than any other single word. Not so with the 'how.' The how inquires into the structure of an event, and once the structure is clear all the whys are automatically answered."*

The Gestalt Approach, page 77

How many 'Why?' questions have you asked in the course of your testing career? Did any of them result in a negative response? Why isn't necessarily a *bad* question to ask, but sometimes we ask, or talk about, 'Why?' When we really mean 'How?'

Simple to Apply



An example requirement annotated using the Meta Model

When reading requirements documentation, I make notes on the requirement documentation and build adhoc summary diagrams. I apply the Meta Model when I go through the document by annotating the document with abbreviations for the Meta Model elements [LP(Lost Performative), RI(Referential Index), D (Deletion), etc.]

I prefer to go through a requirement document several times, and each time I can identify whether my concerns about a Lost Performative are justified (my question may have been answered later in the document). Using the Meta Model results in my reading through requirement documentation faster and producing less long hand notes to represent my concerns.

There is an overlap between the questions in the Meta Mode. When presented with a single statement; there are usually a number of Meta Model questions that are applicable. In order to identify which question is suitable, we have to have in mind the kind of information that we want to obtain. Much like testing, in order to decide what test we want to run, we have to know what we want to know about the system.

There is also overlap between each of the Meta Model elements and arguments can be made for putting them in different classifications e.g. Complex Equivalence could be classified as Deletion if we were to

focus on the equivalence as being deleted from the statement. As you examine the Meta Model elements in more detail you will identify which ones can be placed in different categorisations.

What is important is not the Classification but the impetus behind asking the Meta Model question, i.e. when we identify a Meta Model element what is the information that we want to get from asking the question.

Learn by reading Transcripts

Read transcripts of therapy sessions and apply the Meta Model to those, if you want to understand how the language of therapy works.

But this also applies to the reading of transcripts of testing. Look at your own testing. How did you know to ask that question of the software?

If a question that you asked of software didn't return, what you thought was, any useful information then ask yourself was there any question that would have given you useful information? How would you have known to ask that question? What useful information did you want to get?

Psychology is littered with transcripts. Software Testing is less so. But examine "How to Break Software" probably the closest book that software testing has to a set of transcripts and analyse it. How did they know to ask that question? What would it have taken for you to ask that question? How will you know to ask that question when you are testing?

Learn by Doing

> *"You can't develop a better appreciation of the art merely by reading a book about it. If you want to understand music better, you can do nothing more important than listen to it…All of us, professionals and nonprofessionals, are forever trying to deepen our understanding of the art. Reading a book may sometimes help us. But nothing can replace the prime consideration – listening to music itself"*
>
> Aaron Copland,
> What to Listen for in Music
> 2nd Edition, 1957, McGraw-Hill

Learning the Meta Model is done through application. And since the Meta Model is related to language, all we have to do is listen to people and identify aspects of the Meta Model which apply to their communication. Do this in stages, choose an element, and listen out for it for a day. Listening to the radio, the television or the people you come in contact with.

It is even easier to practise on documentation. You can do it at your own speed:

- Take a document
- Start marking out the Meta Model elements that you find.
    - Ask yourself:
        - Does this contribute to ambiguity in the document?
        - Could this ambiguity lead to problems?
        - What question can you ask to get more information and remove the ambiguity?
        - What tests could you run in the system to check for problems relating to the ambiguity?

The Meta Model is not hard to learn to apply. We apply the Meta Model by taking the statements that are communicated to us literally. Where there is ambiguity we can use as many different interpretations of that ambiguity as possible, if they conflict then it is highly likely that, by chasing the ambiguity, we will identify defects.

References

[SoM1] The Structure of Magic, Volume 1, Richard Bandler & John Grinder, Science and Behaviour books, 1975

[PE95] Persuasion Engineering, Richard Bandler and John LaValle, Meta Publications, 1995

[RD1] NLP Encyclopaedia, Robert Dilts, http://www.nlpuniversitypress.com/

[AK71] Think Like a Grandmaster, Alexander Kotov, Batsford, 1971

[FP1] The Gestalt Approach & Eye-Witness to Therapy, Fritz Perls, Bantam, 1976

[JW66] John B. Watson, Behaviourism, The University of Chicago Press, 1966

I was taught 2nd, 3rd, 4th order presupposition analysis by Eric Robbie and portions of 'Finding THE Test' are adapted from his 'The One Question Exercise'.

Then he shook the rug!
CRACK!
Now the bed had the spot!
And all I could say was,
"Now what, Cat?
NOW What?"

The Cat in the Hat Comes Back,
Dr Seuss

Appendices

The following appendices are designed to be printed out for easy reference outside the context of this paper.

- An Annotated Example
  - Expands upon the presuppositions, and meta model elements examples presented earlier. This shows the depth of ambiguity present in a simple statement and the range of testing that can be performed.

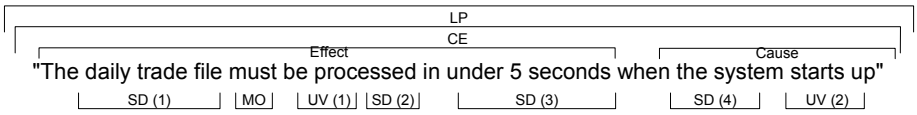
- Meta Model Summary
  - The Meta Model Mind Map from the paper with a table summary of the meta model elements.

- Meta Model Summary Diagrams
  - All 3 Meta Model Summary Mind Maps from the paper on a single.

An Annotated Meta Model Example

"The daily trade file must be processed in under 5 seconds when the system starts up"

Meta Model Elements



LP	Lost Performative
MO	Modal Operator
UV	Unspecified Verb
SD	Simple Deletion
CE	Cause Effect

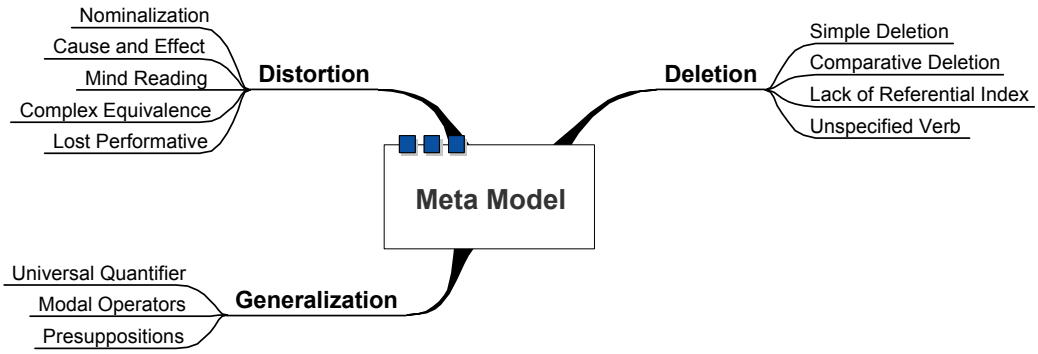
1. Identify the Meta Model Elements
2. What questions can we ask?
3. Identify the presuppositions
4. What do they lead to?
5. Ask the questions early
6. Use the questions as test conditions
7. Use the questions to guide your exploratory testing

LP	Who says "The daily trade file must…"?
SD(1)	Containing what? From where? Called what? Stored where?
MO	What if it isn't?
UV (1)	Processed How specifically?
SD (2)	Processed by what? It might be a mistake to assume it is the same system that starts up.
SD (3)	Where did this number come from? How Much under 5 seconds?
SD (4)	Which system?
UV (2)	Starts how? Starts When?
CE	Processed only when the system starts up? Never processed at any other time? Processed by any other systems? So if the system didn't start it wouldn't be processed?

Presuppositions

Presupposition	Leads to…
There is a daily trade file	what if it doesn't exist?
The trade file is generated in some way	by what system?
The trade file is accessible	is it on our environment, where does it reside? What if the permissions are set wrong?
The trade file can be processed	What if it is corrupt? Locked? Too Big?
The trade file is a file	What kind of file? Binary, Text?)
The system will start-up on a daily basis	What if the system doesn't start up one day, does the trade file get overwritten or something worse? What if the system starts up more than once?
There is only one daily trade file	
The system can process a trade file in under 5 seconds	What if it is too big? What if the transactions are very complicated?
There are knock on effects if the file is not processed in under '5' seconds : 'must'	What happens if it doesn't?
The daily trade file only contains trades for a day?	What does the File contain? What does a trade look like? What is a trade? What kinds of trades?
The system knows how to find the file	Is the location hard coded? Is the location passed as a parameter? Is the location in a config file?
The trade file has a specific name format	Is it always named the same thing?
The daily transaction file has transactions in it	What if the file is empty?
The daily transaction file is produced daily	Produced by what system? Produced when?
The daily transaction file is generated before our system starts up	What if the system starts up as the file is being generated?
The system is closed down at some point (in order to start up again)	What if the system is not closed down?

Meta Model Summary



Meta Model Element	Examples	Questions
Nominalization (N) *A process represented as a noun*	"The Software Testing is inadequate" "Testers don't get any respect"	*Convert object to a process* "What is involved in your software testing?" "How do you know you are not respected?"
Cause & Effect (CE) *X Causes Y* **Stimulus and response**	"Late delivery of software makes me angry"	*Identify missing information associated with the causality* "How specifically does it make you angry?" "Does it always make you angry?"
Mind Reading (MR) *Claim to knowledge (often about someone else's internal state)*	"They hate testers"	**Identify source of claim** "How do you know?"
Complex Equivalence (EQ) *2 statements represented as equivalent*	"They really don't know how to develop software, the requirements are never signed off."	*Check validity of the equivalence* "So if the requirements were signed off, that would mean that they knew how to develop software? Is that all it takes?"
Lost Performative (LP) *A value judgment with a missing judge and criteria.*	"It is wrong to test without writing Test Scripts first"	*Identify Source and criteria* "Who says?", "According to whom?"
Simple Deletion (SD) *Element missing from surface structure.*	"This system is going to fail"	*Recover deleted information.* "Fail how?", "which system?"
Comparative Deletion (CD) *Missing reference point.*	"This system is the worst" "This system is rubbish"	*Identify the unstated comparison.* "Compared to what?"
Lack of Referential Index (RI) **Noun not specified**	"They have a poor process"	*Identify the unstated reference.* "Who are they?"
Unspecified Verb (UV) **Missing process details**	"They want me to test this"	**Identify & expand the unstated verb** "How specifically do they want you to test this?"
Universal Quantifier (UQ) **A Generalization**	"Nobody will ever do that"	**Find a counter example to counter a limiting generalization** "I just did it"
Modal Operator (MO) **Rules for limits**	"We must add this requirement now" "I can't run these tests"	**Explore the rule** "What would happen if you didn't add that requirement now?" "What stops you running these tests?"
Presupposition (P) *What has to be true for this statement to be true?*	"If they know how much work unsigned-off requirements caused us they would make sure they were all signed."	*Challenge presuppositions and test our analysis* "How do you know they don't know?"

Table: Meta Model Elements, examples, questions and objectives

Meta Model Summary Diagrams