Method: Task Breakdown during High level design –*by L. Melcher- 2007 - Version3*

E-Task Definition

Managing Projects by E-tasks breakdowns (atomic or elementary tasks) is an application of the probabilistic law of large numbers: while some tasks will take longer than initially projected, others will take less. It follows that we can:

a) Temporary re-allocate additional resources to late tasks (from the pool of completed parallel tasks, and

b) Re-distribute the overall probability that all tasks will finish according to schedule, as a late task and an early task cancel each other in the overall schedule time estimate.

An elementary task (e-task) cannot be objectively (scientifically) 'justified' nor 'proven' by ordinary deduction.
An e-task is thus a somewhat subjective decision.
An e-task is decided ahead of time and injected into the main project schedule
An e-task should not be over 2 weeks long
An e-task is part of a larger inductive process[1] and the very core of the project schedule and its iterative estimates – *See Bayle's Update Theorem*
An e-task is Axiomatic. They are axioms or part of an a**xiomatic process defined as an inductive process*.** An **axiom** is any starting **assumption** from which other statements are logically derived. It can be a sentence, a proposition, a statement or a rule that forms the basis of a formal system. Unlike theorems, axioms cannot be derived by principles of deduction, nor are they demonstrable by formal proofs—simply because they are starting assumptions—there is nothing else they logically follow from (otherwise they would be called theorems). In many contexts, "axiom," "postulate," and "assumption" are used interchangeably.

How to define an e-task *(Borrowing from Descartes)*

1. **Accept as true only what is indubitable.**
2. **Divide every question into manageable parts.**
3. **Begin with the simplest issues and ascend to the more complex.**
4. **Review frequently enough to retain the whole argument at once**

Project methodology

1. **all complex elements are deemed to be aggregates of simpler ones**

NOTE: Complexity theory or complex systems with infinite variables – think Chaos theory, Traffic, business inner-workings, software development… would totally disagree with #1. Although this discussion is beyond the scope of this document, suffice to say that we consider complexity theory as an ill-formed system with no coherent definition, no scope and no research boundaries. Moreover it seems to rely on two concepts we cannot accept[1]

A) The irreducibility of a complex system

B) The notion of randomness, chance or chaos

The two (A and B) are actually always defined unknowns, yet always explained Randomness may just be a state of limited understanding, and not exist per se.
See L.Wittgenstein in Tractatus Logico Philosophicus

2. **all aggregates(complex elements) are deemed finite**
3. **All complex elements have a beginning (The question: why?) and an end (The answer: This!)**
4. **One aggregate at the time will be de-composed. Step by step into series of simpler elements until it makes no more common sense to break down any further.**
5. **Common Sense is the ability of all trained minds to "sense" a beginning and the end of a proposition. This sense is converted to a certitude, This sense becomes a mathematically true statement (indubitable)**
6. **Break down aggregate until a set of simple "element" can re-create previous aggregate (this will be proof that you have defined the issue correctly) – Regression to the root cause**
7. **Organize elements into pre-low level specification(s)**
8. **Organize element (s) into possible task assignment to self or to third party**
9. **Organize element (s) into time frames**
10. **look over all aggregates and make sure they are all fully distilled**
11. **review once more if you can distill more**
12. **This will reveal how many man-hours you need, and the percent of work to be done by each man.**

Example: Project Phase: "High Level Specification / Code Design Phase Start"

One method will allow me to schedule

- Senior Systems Architect - Programmer 1
    - (option one) Depending on the resulting task processed by the e-task method, he will be allocated to two parallel tasks:
    - Supervise, document and review (with 1 additional System Architect) high level specifications and write guidelines for low level coding (C#/ASP.net) for 1 month + ½ at 40 % of his time
    - Program 60% of his time in VXML and WSDL (web service interface contract)
    - (option two) His time, depending on result processed by the method will be divided into:
    - 25 percent allocated to coding and 75 percent working alongside lower level junior C# programmers.
- Web / Interface design

Exceptions: Architectural Backbone – led by Critical Path

- Programmer 2 – Senior Architect, C, C++ Programmer
    - 80 % for 1.8 months on the 'Pre-fetch engine' task or what we would call the main architectural process or 'core mechanism' in the project's architecture; this is a situation to avoid at all cost. Modularity and reconfiguration are keys to success.
    - NOTE: Any low level process that supports the software / hardware architecture that cannot / will not be shared or broken down into multiple e-tasks, that is, if no other solution can be found and provided the E-Task breakdown method has bee applied to this aggregate and proven that the task cannot be partially de-allocated - is eventually going to be a death sentence. Down the line as upgrades, patches and customizations will be needed or requested, these will make this indivisible task grow exponentially into the realm of impossibility.
    - The other 10 % would go to project maintenance, architecture review and tune-up and code review.

- If we go thru high-level specifications in a methodical way – as described in this document - by disassembling data aggregate into ALL of their parts.
- If we break all complex programming units into a lot of simpler algorithms and functions,
- Then we can do something that would save a great amount of time. We could deduct from **high-level** specs most of the data needed for **low-level** specifications and
- be able to create task assignments and approx duration of tasks at the same time:

We can postulate the following:
The overall cost (**without contingency planning**) of an etask is thus:
et = t(tc) x r(nc) where an Etask is the sum of itself: t(tc) and nc is a variable cost per hour (n) of a resource - programmer (r) – n being based on seniority level.
➔ Cost of sub-project (phase 1)
P(1) = Σ (r(nc) x t(x)), where t(x) is the total amount of tasks (x) in Project - Phase 1
➔ The sum of all contingencies should be accounted and evaluated in the range of <35% -> 50%> and multiplied by existing cost analysis.

NOTES
* **Brief definition of induction:** Probable reasoning whose conclusion goes beyond what is formally contained in its premises.
1 - This process is a subset of a methodology using conditional logic and Markov models: Quantifications and attributions of events are conditional probabilities based on degrees of confirmation or Criterion of Adequacy (CoA) or degrees of confirmation of an unknown event B inferred from a known event A (Prb(A|B) (See Paper: '*Probability Factors, the future and Predictions*'
2- See paper on Complexity Theory: '*Complexity Theory or the Theory of the void*")