Ideen und Stichpunkte zu Trial&Error
(Stand von Mitte Juni 2006)

WAS IST TRIAL AND ERROR BZW. WAS KANN TRIAL AND ERROR ALLES SEIN (ALLES HYPOTHESEN!!):

Abkürzung: Trial&Error = T&E

- Menschen neigen dazu, nicht funktionierende Dinge redundant zu versuchen.
  Bsp1: Existiert eine bestimmte Variable (noch) nicht, so wird sie von Eclipse unterkringelt. Der Benutzer möchte dies nicht wahrhaben, löscht den Namen und gibt ihn erneut ein, wobei das Ergebnis natürlich das gleiche ist.
  Bsp2: mehrmaliges klicken auf einen Link, wenn er nicht funktioniert.

- Programmierer hat nicht den absoluten Überblick über den Code, weshalb er scheinbar wahllos Dinge probiert. Ein gutes Beispiel stellen verschachtelte Schleifen dar, welche natürlich jeweils abgeschlossen seien müssen. Wenn nicht geeignet eingerückt, so ist meist unklar, wie viele Schleifen noch zu schließen sind. dies wird einfach getestet, bzw. so lange probiert, bis die Entwicklungsumgebung keine Fehler mehr anzeigt.

- Problem: verschachtelte Ausdrücke, wie etwa mathematische Formeln oder komplizierter Objektorientierter "Spaghetticode" (d.h. lange Zeile mit vielen Methoden-Punkten). Klammersetzung nicht klar. Es wird wahllos probiert, bis alles klappt. Häufig wird dieser Ausdruck seitens des Programmierers keinem kompletten "Parsing" unterzogen, in welchem sich der Programmierer die Struktur des Ausdrucks noch einmal klar machen würde.
  WICHTIG: solch ein kompletter Überblick erfolgt erst nach einigen erfolglosen Klammerverschiebungsaktionen.
  Bsp: MCAM-Video "*ACM050621_A6_0.06.08.54_0.23.38.10*" ab Minute 05.00 -> erfolglos versucht, Java-Programm zu starten wegen "*-cp .*"

- Menschen scheinen Angst vor zu großen Schritten zu haben.
  Bsp: (GUI-Entwicklung) es wird lange die richtige Position für einen Button gesucht. Dieser wird immer nur um ein paar Pixel verschoben und das Resultat danach betrachtet. Man könnte den Knopf auch gleich sehr viel weiter verschieben, aber das wird nur selten gemacht.
  Weitere Beobachtung: je näher sich Menschen dem Ziel zu nähern glauben, desto vorsichtiger werden sie (d.h. die Änderungen werden immer kleiner) und desto mehr Zeit vergeht zwischen zwei Kompilations-Vorgängen, weil Menschen langsamer arbeiten, selbst wenn das Ergebnis/Optimum noch lange nicht erreicht ist.

- speziell in Entwicklungswerkzeugen wie Eclipse:
  - auf Autovervollständigen hoffen: die ersten Lettern einer Variable/Operation werden eingegeben und das Passende ergänzt.
  - auf „CTRL+Space" vertrauen -> automatische Ergänzung der Variablen/Methode
  - unbedarft einen Dateityp wählen oder Klassen, welche noch nicht importiert wurden und auf die Fehlersymbole am linken Rand hovern und schauen, was der - bewusst herbeigeführte - Fehler war; ggf. von Eclipse selbst beheben lassen (wie

etwa eine fehlende klasse zu importieren.)

Bedeutung/Abstraktion:
Auslagern von Informationen auf den Rechner: durch geschickte Kombination an wenigen Schlagworten lassen sich Informationen beschaffen, da ein Computer diese geeignet zu verknüpfen mag.
Diese Methodik findet zunehmend nicht nur in der Programmierung statt sondern auch bei normalen Anwendungen, insbesondere bei der (Online-/Desktop-)Suche: Menschen haben sich nur einen Bruchteil der notwendigen/gewünschten Information eingeprägt, welche dem Computer aber ausreicht, um die volle Information zu beschaffen.
Bsp.: (Suche nach Stichwörtern) Mensch liest Artikel im Internet -> verlässt die Seite und merkt sich URL nicht -> möchte später auf Artikel wieder zugreifen -> hat sich einige wenige Schlagworte memoriert, welche es Suchmaschinen erlauben die URL zu finden.

ERSTES FAZIT:

- Zwei Kategorien von T&E
  - bewusst (als Methodik)
  - unbewusst
- Ein Großteil von T&E findet nicht in Interaktion zwischen Code und Benutzer statt, sondern zwischen Code und der Entwicklungsumgebung
  - Entwicklungsumgebung bildet mehr und mehr eine neue Schicht zwischen Code und Benutzer
  - viele, früher oft genutzte T&E-Fälle existieren heutzutage nicht mehr, da die Entwicklungsumgebungen dem Entwickler diese Probleme abnehmen

---

T&E ermöglicht es, viele Möglichkeiten Durchzuspielen (anders formuliert: viele Chancen wahrzunehmen), wobei die most-likely Möglichkeit zum Herbeiführen der gewünschten Lösung als erstes probiert wird.

Dadurch, dass bei T&E die Eingabemenge stark begrenzt ist (z.B. bei einer Suche nur wenige Schlagwörter einzugeben sind), sind viele Chancen in kurzer Zeit möglich. zudem können diese vielen Chancen/Schritte eine allmähliche Annäherung an das Ziel bewirken. T&E entspricht also dem Verfahren einer allmählichen Approximation an ein gewünschtes Ziel.
Gleichzeitig können die nicht erwünschten/nicht optimalen Ergebnisse leicht verworfen werden, da sie leicht herbeizuführen/ohne bzw. ohne großen Ressourcen-/Zeiteinsatz erreichbar waren.

HIER RELEVANTE ARTEN VON T&E

In dieser Arbeit wird nur semantisches T&E betrachtet.
Semantische T&E-Episoden lassen sich als Episoden definieren, welche T&E-Vorgänge erst ab dem Zeitpunkt betrachten, ab dem der Code ausführbar ist, also frei von syntaktischen Fehlern ist . Eine Semantische T&E-Episode ist eine Phase des Programmierens, in welcher der Programmierer die inhaltlichen Fehler seines Programms entfernt/versucht zu entfernen, wobei er aber keine wohlüberlegten Codeänderungen tätigt, sondern den Code wild abändert und so lange probiert und rätselt, bis alle Fehler beseitigt zu sein scheinen.

NOTWENDIGE GRANULARITÄTEN BEZÜGLICH CODEÄNDERUNGEN

Welches Detailbetrachtungsniveau bezüglich des Codes ist notwendig, um Episoden eindeutig zu erkennen:

1) Datei-Niveau
   - nicht fein genug, bzw. falsches Modell
2) Klassen-Niveau
   - wichtig, um Codeabhängigkeiten zu beobachten
   - ist alleine aber zu grob
3) Methoden-Niveau
   - siehe 2)
4) Zeilen-Niveau
   - noch zu grob da nicht unterschieden kann:
     - welcher T&E-Fall war es
       - z.B. war es Auskommentieren von Code oder eine Schleifenänderung (Details s.u.)
     - unterschiedliche Episoden besitzen unterschiedl. Merkmale bezogen auf Änderungshäufigkeit, d.h. die Anzahl der Änderungen pro Zeile ist wichtig
     - evtl. keine Eindeutigkeit, ob betrachtete Episode noch aktuell ist oder ob sich momentane Änderung dieser Zeile bereits auf anderen Kontext bezieht
5) Zeichen-Niveau
   - optimal, aber sehr schwierig zu implementieren!!

folgendes ist also notwendig, um 5) gerecht zu werden:

- Zeileninhalt (also jedes einzelne Zeichen) muss vom ECG übertragen werden
- Bestimmung, ob sich ein ausgeführtes „run/debug" auch auf die aktuell beobachtete T&E-Episode bezieht
  - bzw. Bestimmung, ob sich das „run/debug" auf die zuletzt geänderte Zeile bezogen hat
- Methode/Algorithmus notwendig, welcher auf Ähnlichkeiten/Unterschiede zwischen verschiedenen Versionen ein und derselben Zeile prüft
  - z.B. in der Form: „20%ige Veränderung der Zeile von Version x zu Version y"
  - „Argument einer for-Schleife von x auf y geändert"

Vereinfachungen, um Granularität zu erhöhen:

- nur die ersten Zeichen einer Zeile werden betrachtet, ob es sich um einen Kommentar handelt
  - Problem: um Sprachunabhängigkeit zu gewährleisten müssen die Kommentarzeichen generisch sein (z.B. „//" für Java, aber „--" für Haskell)
- Zähle nur die Anzahl der Zeichen der Zeile pro Zeilenversion
  - Kontra: bei Schleifen ist Änderung von „0" auf „1" die Änderung der Länge der Zeile gleich Null!
  - Pro: so lassen sich leicht Episoden wie „Ersetze komplexe Ausdrücke durch Einfache „ erkennen -> evtl. nur Codeänderungen mit Distanz größer als 4 oder so zählen
  - aber SEHR großes Problem:
    - Identifikation, wann sich eine Zeile noch um dieselbe Zeile handelt
    - aka: wann wird eine Zeile verschoben/gelöscht?
      - Erkennen ist äußerst komplex
        - und sei es nur, dass Programmierer eine Leerzeile einfügt

- Lokalisierung der relevanten Zeile(n) ist schwer

weitere Probleme:

- Programmier tätigen während T&E-Phase kleine Änderungen am Code, auch wenn diese Änderungen für die Phase irrelevant sind, diese also nicht unterbrechen
  - Bsp: Schnell mal eben Leerzeilen einfügen oder etwa kommentieren

Realität:

- das ECG liefert keine Zeichengranularität!!!
- möglicherweise Einsatz eines Algorithmus von Sebastian Jekutsch, welcher Blöcke erkennt
- also lassen sich in Praxis die definierten T&E-Episoden nur sehr viel grobkörniger implementieren, liefern also SEHR viel gröbere Ergebnisse...
- von Sebastian und mir vereinbarte Einschränkungen
  - keine Betrachtung von Codeabhängigkeiten zwischen Methoden, sondern nur innerhalb eines Blocks
  - kein Parsing von Eclipse des Java-Codes durchführen lassen

---

TRIAL&ERROR VS. DEBUGGING

Was ist der Unteschied zwischen T&E und Debugging.

- Eine mögliche Abgrenzung:
  - Mit T&E-Vorgehen geht gezwungendermaßen einher, dass der Programmierer Codeänderungen tätigt. Ausnahme: Das Einfügen von Bildschirmausgaben und dergleichen zum Programmverständnis.
- Oft ist Debugging nichts anderes als Codeverständnis

GENERELLES VORGEHEN BEI INTERVIEWS UND OBSERVIERUNGEN:

Dauer: ca. 3h + Pause
davon: - 1/2 zu Beginn Einführungsdiskussion
- 2h Beobachtung beim Programmieren
- Pause (15min)
- 1/2 Schlussdiskussion

- Einführungsdiskussion:
  - zu Beginn: Probanden fragen, was er unter T&E versteht.
  - Erklärung meinerseits, was T&E in meinen Augen ist.
  - dem Probanden klar machen, dass ein Großteil von T&E durch Interaktion mit Entwicklungsumgebung entsteht
  - sich vom Probanden seine Entwicklungsumgebung erklären lassen sowie seine Kenntnisse in derselbigen.
  - dem Probanden bitten, selbst auf evtl. T&E-Episoden zu achten
- Observierung:
  - IM VORFELD FORMULAR ERSTELLEN!!

- aktuelles Datum
- Name des Probanden
- Anfangszeit, Endzeit, Dauer
- Auflistungen der T&E-Typen (für die Strichliste)
- leere Felder für beliebige Bemerkungen

- Anfangszeit aufschreiben
- Zeiten vermerken, wenn ich T&E bemerkt habe
- Welches T&E habe ich bemerkt (Strichliste)
- welche T&E hat Proband selbst bemerkt?
- sonstiges

ANALYSE VON INTERVIEWS UND OBSERVIERUNGEN:

- Alle Formulare analysieren.
- evtl. T&E-Episoden verwerfen, welche nicht vorkamen, also keine Striche auf der Strichliste haben
- Welche Unterschiede gibt es zwischen erfahrenden und nicht erfahrenden Programmierern (ist das überhaupt wichtig und gehört noch in den Bereich meiner Arbeit?)

FRAGEN + PROBLEME:

- In welche Kategorie lassen sich Vorgänge einordnen wie: *Programm Laufen lassen -> testen und evtl. bewusst Fehler erzeugen -> diese Fehler danach analysieren*. Ist das auch eine Art T&E? Mache erst alle Fehler und schaue danach die Ursachen dafür nach? (Bin gekommen auf diese Idee durch das "*Tolksdorf050926_P4_4_15.08.00+1.08.12*"-Video)

- Letztendlich lässt sich sehr viel in die Kategorie/Methodik T&E eingliedern. Beispielsweise ist eine Suche in der Java API nach einer Klasse/Operation, welche ein gewünschtes Ergebnis liefern soll ach T&E, da der Benutzer zuerst in den Klassen bzw. nach den Begriffen sucht, die ihm am erfolgversprechendsten aussehen. Dieser Vorgang einer Suche ist aber eine im Leben alltägliche Tätigkeit. Ist also das einzige was der Mensch macht, T&E???