# SOLUSI DALAM MENGHADAPI KETERBATASAN WAKTU SELAMA PROSES PENGUJIAN *SOFTWARE*

**Nia Gella Augoestien** [1)]**, Azhari SN**[2)]
[1,2)] Jurusan Ilmu Komputer dan Eletronika FakultasMIPA Universitas Gadjah Mada
Sekip Utara, Bulaksumur, Yogyakarta 55281
e-mail : nia.gella@mail.ugm.ac.id, arisn@ugm.ac.id

**Abstrak**
*Pengujian merupakan tahap penting dalam proses pengembangan software, sekaligus merupakan aktifitas paling mahal dalam tahap pengembangan software. Salah-satu tujuan utama dari pengujian adalah untuk dapat mengungkapkan sebanyak mungkin kecacatan pada software dan memperbaikinya. Sedangkan tujuan lain adalah untuk mengetahui software telah memiliki spesifikasi sesuai yang diharapkan. Ada berbagai macam teknik dan strategi yang dapat digunakan selama proses pengujian. Namun pengujian sering dihadapkan terhadap keterbatasan waktu pengembangan suatu software agar dapat memenangkan persaingan pasar. Sehingga diperlukan strategi khusus untuk melaksanakan pengujian dalam waktu yang singkat, tetapi tetap menghasilkan software dengan kualitas baik . Dengan mengkombinasikan teknik fault-based dan fungsional diharapkan mampu memberikan strategi baru untuk memaksimalkan deteksi kecacatan yang terdapat pada software selama masa pengujian.*

***Kata Kunci** : Pengujian software, Pengujian fungsional , Deteksi kecacatan*

## 1. PENDAHULUAN

Pengujian merupakan tahap penting dalam setiap proses pengembangan *software*. Hal ini dicerminkan dengan keberadaan proses pengujian dalam setiap metode pengembangan *software*, mulai dari metode formal sampai metode non-formal seperti *extreme progamming*. Sangat sulit untuk mengimajinasikan implementasi aplikasi atau rilis baru tanpa proses pengujian. *Software* yang memiliki kualitas yang baik adalah produk *software* yang menyediakan nilai kepuasan penggunanya, membuat keuntungan dan menghasilkan sedikit keluhan (Sibgatullina, 2011). Selain itu *software* dituntut untuk dikembangkan dalam waktu yang sangat singkat untuk dapat memenangkan persaingan pasar.

Menurut Sommerville (2009) Pengujian dimaksudkan untuk menunjukkan bahwa program melakukan fungsinya dan menemukan cacat pada program sebelum suatu program benar-benar digunakan. Sedangkan menurut Machado (2010), pengujian *software* adalah kegiatan dinamik yang membutuhkan produk untuk dieksekusi dengan input yang diberikan dan menghasilkan output yang dapat diobservasi. Pengujian *software* dikenal sebagai aktivitas paling mahal dalam proses pengembangkan karena mengambil 50% dari biaya total proyek *software*, sehingga sangat masuk akal dengan memberikan perhatian lebih pada proses pengujian akan membawa keuntungan yang lebih dalam proses pengembangan *software*. Dan tidak jarang proses pengujian yang tepat yang akan mampu menghasilkan produk *software* yang memiliki kualitas lebih baik.

## 2. TINJAUAN PUSTAKA

Ada beberapa beberapa aspek yang harus diperhatikan sehingga didapatkan proses pengujian yang efektif dan efisien sehingga tuntutan dalam melaksanakan pengujian dalam waktu yang singkat dapat dipenuhi, dan disisi lain dapat menghasilkan *software* dengan kualitas yang lebih baik. Adapun aspek yang perlu diperhatikan tersebut adalah

- Tujuan pengujian
- Aktivitas utama pengujian
- Keterbatasan pengujian

### 3. 1. Tujuan Pengujian

Dengan fokus terhadap tujuan dari pengujian maka dapat tentukan teknik dan strategi yang tepat untuk pengujian yang akan dilakukan. Menurut Sommorville (2009), terdapat 2 tujuan jelas dalam proses pengujian, yaitu;

- Untuk menunjukkan pada pengguna, *software* yang dibuat sesuai dengan kebutuhan. Untuk *software* yang dibuat menurut pesanan, hal ini berarti harus ada paling tidak satu test untuk setiap persyaratan pada dokumen persyaratan. Sedangkan untuk produk *software* yang umum, berarti harus terdapat pengujian untuk semua fitur yang ditawarkan yang ditawarkan sistem dan kombinasi dari semua fitur tersebut yang menyatu dengan rilis produk.
- Menemukan situasi dimana prilaku dari *software* yang tidak semestinya, tidak diinginkan, dan tidak cocok dengan spesifikasi, yang merupakan konsekuensi dari kecacatan *software*. Pengujian kecacatan difokuskan

pada membasmi prilaku yang tidak diinginkan, seperti sistem crash, interaksi yang tidak diinginkan dengan sistem lain, komputasi yang salah, dan kerusakan data.

Tujuan lain dari pengujian untuk menemukan bugs dan meralatnya, Kumar dan Rana(2011). Metode pengujian berbeda akan menghasilkan tujuan yang berbeda. Senada dengan pendapat Machado (2010) tujuan utama dari pengujian adalah mengungkapkan kecacatan yang terdapat pada produk yang diuji selama fase-fase pengembangannya. Menurut Nair (2008) Terdapat 3 ruang lingkup yang mencakup kecacatan pada *software* yaitu;

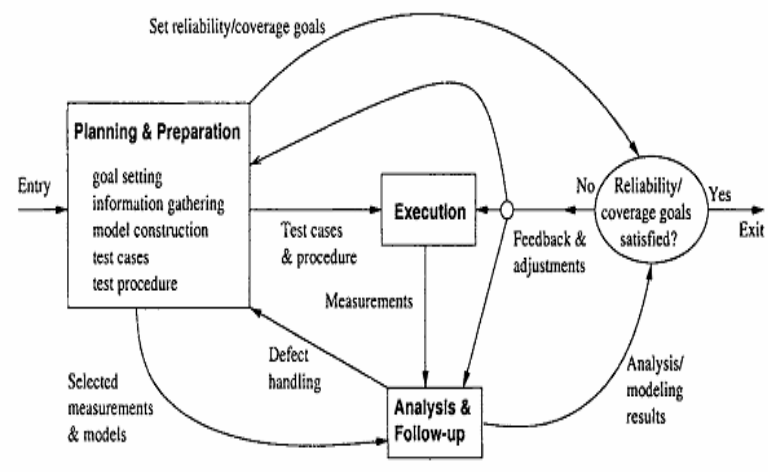
- Error : Aksi kegiatan manusia yang membawa ke hasil yang tidak benar.
- Fault : keputusan yang tidak benar tang diambil ketika memahami informasi yang diberikan pad implementasi proses.
- Faiure : Ketidak mampuan *software* berfungsi sesuai dengan persyaratan.

Walaupun demikian, dengan adanya proses pengujian pada suatu sistem yang dikembangkan, tidak berarti secara otomatis sistem yang dihasilkan akan berfungsi sesuai dengan yang diharapkan atau dapat bebas dari kecacatan. Menurut Machado (2010) sangat tidak mungkin untuk memproduksi *software* yang benar-benar bebas dari cacat. Hal ini dikarenakan "*Testing can only show the presence of errors, not their absence*" (Sommorville,2009). Agar dapat meminimalisir kecacatan yang tidak ditemukan pada proses pengujian maka diperlukan stategi yang tepat dalam proses pengujian. Dengan tujuan akhir agar *software* memiliki kualitas yang baik.

Terdapat 3 pengukuran utama untuk kualitas suatu produk: biaya, lingkupan, dan waktu (Sibgatullina, 2011). Suatu *software* yang baik hanya membutuhkan biaya yang sedikit dan waktu yang pendek dalam pengembangannya, namun dituntut mampu memenuhi standart kebutuhan pengguna. Waktu pengujian yang lama dapat menyebabkan terlambatnya jadwal rilis suatu *software*, namun dengan waktu pengujian yang relative singkat tidak menjamin suatu *software* dapat berfungsi sesuai dengan yang diharapkan.

### 3. 2. Aktivitas Utama Pengujian

Sebagai salah satu fase dalam pengembangan *software*, pengujian memiliki rangkaian aktifitas utama dalam prosesnya. Aktifitas-akfitas tersebut dimaksudkan agar tujuan dari pengujian *software* itu sendiri tercapai. Secara garis besar dikelompokkan menjadi 3, yaitu; perencanaan dan persiapan, eksekusi pengujian, analisa dan perbaikan (Tian, 2005), seperti yang terlihat pada Gambar 1.




**Gambar 1.** Rangkaian Aktifitas Pengujian

Terdapat level yang berbeda dari pengujian yang berhubungan dengan level abstraksi pada code, yaitu:

- Level yang paling detail → elemen-elemen dari program diuji secara individual, termasuk didalamnya, pernyataan-pernyataan pada program, percabangan, item data yang biasayan focus pada ruang lingkup yang kecil yaitu unit, yang biasanya disajikan dalam bentuk, fungsi, prosedur, method , subrutin dan sebagainya.
- Level menengah →beberapa elemen program diperlakukan sebagai group yang terkoneksi satu sama lain untuk diuji.
- Level yang paling abstrak →keseluruhan sistem *software* diperlakukan seperti black box, dimana focus pada fungsional sistem atau relasi antara input dan output yang dieksusi pada sistem.

**3. 3.Keterbatasan Pengujian**

Menurut Farooq (2010) terdapat beberapa keterbatasan pada proses pengujian *software* yaitu

a. Pengujian hanya dapat digunakan untuk menampilkan kehadiran kesalahan, namun tidak menunjukkan ketidak hadirannya. Pengujian tidak mampu mengindikasi keberadaan kecacatan pada *software* yang belum terdeteksi olehnya. Jadi setelah melakukan pengujian tidak ada garansi *software* bebas dari cacat.
b. Pengujian tidak akan membantu ketika kita diharuskan untuk mengambil keputusan apakah harus merilis produk dengan masih banyak kesalahan demi mengejar target, atau terlambat merilis produk dengan kecacatan yang lebih sedikit demi kualitas *software* yang baik.
c. Pengujian tidak dapat mengaransi *software* dapat berfungsi seperti semestinya dalam semua kondisi, namun hanya menunjukkan bahwa *software* tidak dapat berfungsi seperti semestinya pada kondisi-kondisi khusus.
d. Pengujian tidak membantu dalam menemukan akar penyebab kecacatan yang ditemukan. Hanya saja lokasi penemuan cacat pada *software* yang ditemukan sekarang dapat mencegah kecacatan lain dimasa yang akan datang yang ditimbulkan akibatnya.

Karena keterbatasan-keterbatasan ini diperlukan suatu batas yang mentukan kapan pengujian terhadap suatu *software* berhenti dilakukan.Menurut Tian (2005) secara umum, dalam fase pengembangan *software* terdapat 2 hal yang perlu dipertimbangkan untuk memutuskan apakah pengujian terhadap suatu produk telah cukup atau tidak, yaitu:

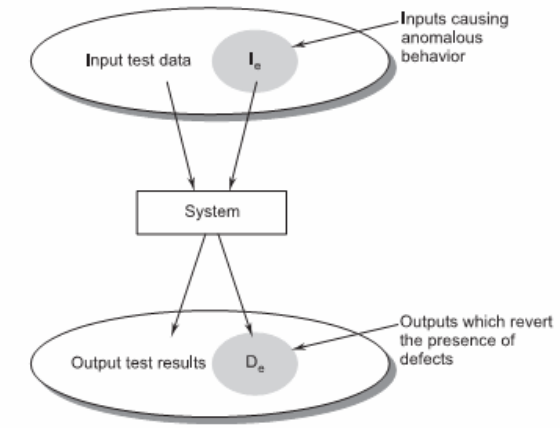
- Berdasarkan kriteria resource yang dimiliki
  Kebanyakan *software* akan terhenti pengujiannya ketika waktu dan biaya yang anggarkan terhadapapnya telah habis. Dimana jadwal rilisi produk dan biaya menjadi suatu batasan yang dominan dalam pengembangan *software*.
- Berdasarkan kriteria aktivitas.
  Diawal proses pengujian terdapat subfase perencanaan proses pengujian. Pada subfase tersebut dirumuskan aktivitas-aktivitas pengujian yang harus dilakukan dalam kaitannya dengan kualitas dari *software*. Pengujian akan berhenti ketika semua aktifitas tersebeu telah terlaksana.

**3. METODE PENELITIAN**

Ada beberapa teknik yang sering digunakan dalam prose pengujian *software*. Dengan mengkombinasikan 2 buah teknik yang ada dan melakukan beberapa pengujian, didapatkan strategi yang mampu memberikan solusi dalam melaksanakan proses pengujian *software* dalam waktu yang terbatas.

**3. 1. Pengujian Fungsional**

Pada teknik pengujian fungsional ini struktrur program dari *software* tidak dipertimbangkan. *Software* dipandang sebagai satu kesatuan yang terintegrasi satu sama lain sehingga dapat diabaikan. Kasus uji yang digunakan selama proses pengujian ditentukan berdasarkan pada spesifikasi yang seharusnya dimiliki oleh *software* (Agarwal, 2010). Dengan fokus pada kelakuan eksternal sistem dan spesifikasi yang dimilikinya, maka *software* diperlakukan seperti sebuah *black-box* (kotak hitam). Pengujian *black-box* dapat dilustrasikan seperti Gambar 2. Bentuk sederhana dari pengujian *black-box* adalah dengan memulai menjalankan *software* dan melakukan observasi membedakan antara perilaku yang diharapkan dengan yang tidak diharapkan.




**Gambar 2**. Pengujian Fungsional

Bentuk lain dari *black-box* adalah menggunakan daftar spesifikasi *software* yang mencatat semua fungsi eksternal yang terdapat pada *software* dan juga beberapa informasi tentang kelakuannya atau pasangan input dan outputnya. Namun cacat yang ditemui pada *software* ketika proses pengujian akan sangat sulit ditemukan

penyebabnya, karena data yang diperoleh dari hasil pengujian hanyalah korelasi antara input dan ouput yang mengindikasi kecacatan.

Pengujian *black-box* dapat mengidentifikasi beberapa macam kecacatan yang terdapat pada *software* sebagai objek uji, yaitu;

- Kecacatan Fungsi
- Kecacatan antarmuka
- Kecacatan model data
- Kecacatan pada akses eksternal sumber data.

Sehingga pengujian ini sering kali dilakukan pada akhir dari proses pengujian, dengan asumsi bahwa teknik sebelumnya pada proses pengujian telah mampu menangani sebagian besar kecacatan lain pada *software* produk.

**3. 2. Pengujian Fault-Based**

*Fault-based* atau pengujian berdasarkan kecacatan program bertujuan untuk memilih deretan kasus uji untuk mendeteksi kemunggkinan-kemungkinan kecacatan pada *software* (Lau, 2007). Menurut Vincenzi (2010). Teknik pengujian *Fault-based* menggunakan informasi tentang kecacatan yang sering ditemukan pada pengembangan *software* dan tentang tipe spesifik cacat yang dinginkan untuk diuji. Jadi pada awal proses pengujian dilakukan estimasi kemungkinan bagian-bagian pada *software* yang memiliki kecacatan. Dengan fokus pada hasil estimasi tersebut dilakukan pengujian untuk membuktikan apakah *software* benar-benar memiliki cacat pada bagian yang dimaksut atau tidak. Pada teknik ini terdapat 2 kriteria yang menjadi initisari pada kecacatan ,yaitu

- *Error seeding*
  Menaburkan estimasi kecacatan-kecatan yang mungkin terjadi pada *software*. Teknik ini didasari pada asusmsi bahwa kecacatan terdistribusi secara seragam pada sebuah *software*.
- *Mutation testing*
  Menginjeksi *software* dengan kecacatan pada source code, lalu menjalankan kasus uji yang dapat mendeteksi kecacatan yang dirancang dengan asumsi kecacatan - kecacatan yang sama akan terdeteksi oleh kasus uji tersebut. Dan jika pada saat eksekusi kasus uji didapat banyak kecacatan pada *software*. Maka mengimplementasikan kasus uji dapat mengefisienkan proses pengujian.

Dengan mengimplementasikan teknik *Fault-based* pada pengujian *software* memungkinkan untuk menemukan sebanyak mungkin kecacatan pada *software*. Namun temuan kecacatan yang maksimal tidak menjamin bahwa *software* telah memiliki spesifikasi sesuai dengan yang diharapkan.

**4. HASIL DAN PEMBAHASAN**

Ada beberapa strategi yang dapat digunakan untuk memperoleh *software* dengan kualitas baik dengan keterbatasan resource dan waktu yang diberikan.

a. Menggunakan kasus uji yang telah digunakan sebelumnya untuk *software* yang sedang dikembangkan.

Kebanyakan dari *software* yang ada saat ini merupakan pengembangan dari *software* yang sudah ada. *Software-software* yang baru dirilis merupakan hasil penambahan spesifikasi dari *software* yang sudah dirilis sebelumnya. Kasus-kasus uji yang digunakan pada pengujian *software* sebelumnya dapat digunakan untuk pengujian *software* yang baru. Hal ini tentu saja dapat mempersingkat waktu pengujian dibandingkan harus mendesain kasus uji yang baru. Sedangkan untuk pengujian spesfikasi baru yang akan ditambahkan dapat merancang satu kasus uji baru saja. Penambahan kasus uji baru bertujuan untuk menanggulangi aspek-aspek produk yang tidak tercakup oleh kasus-kasus uji yang telah ada.

Stategi ini dapat dengan baik digunakan, ketika pengembang *software* mendokumentasikan setiap kasus uji yang telah digunakannya dengan baik. Sehingga kasus-kasus uji dapat didata berdasarkan kecacatan-kecacatan yang dapat ditemukan pada *software* ketika mereka dieksekusi.

b. Menentukan prioritas dari deretan kasus uji yang ada berdasarkan banyaknya kecacatan yang diperoleh ketika menjalankan kasus uji terhadap *software*.

Dengan keterbatasan waktu dan resource dalam pengembangan suatu *software* sulit untuk melakukan eksekusi untuk setiap kasus uji yang ada. Namun dengan memprioritaskan pelaksanaan eksekusi kasus uji berdasarkan banyaknya kecacatan yang ditemukannya pada *software* dapat memaksimalkan temuan kecacatan pada *software* dengan waktu dan resource yang terbatas. Karena pada kenyataannya salah satu tujuan dari pengujian adalah menemukan sebanyak mungkin kecacatan yang terdapat pada *software* dan memperbaikinya.

Secara sederhana dapat dilustrasikan dengan contoh berikut;

*Dari hasil pengujian software pada rilis sebelumnya didapatkan data dari dokumentasi pengujian. Dari eksekusi 5 buah kasus uji yang ada didapat 8 buah kecacatan yang disajikan dalam bentuk Tabel 1 berikut.*

*Tabel 1. Dokumentasi Pengujian Kasus Uji*

| *Kasus Uji\ Kesala-han* | *E1* | *E2* | *E3* | *E4* | *E5* | *E6* | *E7* | *E8* | *Waktu* |
|---|---|---|---|---|---|---|---|---|---|
| *K1* | * | * | | | * | * | | * | *12 ms* |
| *K2* | * | | * | * | * | * | | | *8 ms* |
| *K3* | | * | | * | | * | * | | *8 ms* |
| *K4* | | | * | * | | | * | | *6 ms* |
| *K5* | | * | | | * | | | * | *5 ms* |

*Dengan memprioritaskan kasus uji K1 untuk dieksekusi pada pengujian software kita dapat ditemukan 5 kecacatan (E1, E2, E5,E6, E7,E8) pada software dalam waktu 12ms dan kecacatan E3,E4,E7 dapat ditemukan dengan mengeksekusi kasus uji K4 dalam waktu 6ms. Dengan demikian tidak diperlukan melakukan kasus uji yang lain untuk mendeteksi kecacatan yang sama, sehingga dapat menghemat waktu pengujian 25ms. Karena pengujian yang seharusnya dilakukan 43 ms dcukup dilakukan dalam 18ms saja.*

c. Menggunakan pendekatan pencegahan cacat

Kecacatan pada suatu produk *software* dapat membawa pada situasi yang berbahaya pada setiap fase pengembangan *software*, Nair (2008). Pencegahan cacat merupakan proses mengidentifikasi cacat dari akar penyebabnya dan membenarkannya serta mencegat terjadi kecacatan kembali. Semakin lama suatu cacat terdapat pada suatu produk, semakin sulit kecacatan tersebut dibenarkan. Pencegahan sejak awal tentu akan mengurangi dampak keberadaan kecacatan. Terdapat banyak metode yang dapat digunakan dalam mencegah kecacatan pada *software*, dan Inspeksi *software* dapat digunakan sebagai pendeteksi kecacatan yang palin baik (Nair, 2008).

d. Menggunakan produk analisis statis

Produk analisis statis menguji kode *software* tanpa harus mengeksekusi *software* tersebut. Dengan mengaplikasikan sekumpulan aturan untuk dapat mengidentifikasi kualitas kode selama awal masa pengembangan. Dengan menggunakan produk analisis statis ini dapat membantu mengenali dan membasmi cacat yang terdapat pada *software* sebelum digunakan untuk tahap selanjutnya. Menurut Brisky (2008) banyak studi yang menggunakan produk analisis static mampu menemukan 5 – 30% kecacatan code *software* pada awal masa pengembangan. Ketika kecacatan ditemukan pada awal fase pengembangan, dapat mengurangi sejumlah kecacatan pada fase pengujian yang secara otomatis dapat menghemat waktu dan biaya pengembangan.

e. Mengunakan pengujian black – box sebagai acceptance test

Acceptance test merupakan pengujian yang menetukan apakah suatu produk *software* dinyatakan dapat dirilis atau tidak. Ketika suatu *software* dinyatakan selesai diuji maka segera setelah itu, *software* tersebut akan dirilis. Untuk memberikan bukti bahwa spesifikasi pada *software* telah sesuai harapan costumer, maka dilakukan black box testing. Pada proses ini semua daftar spesifikasi yang harus dimiliki *software* diuji sekaligus diobservasi perilaku *software* apakah sesuai dengan yang diharapkan atau tidak. Korelasi antara output yang dihasilkan *software* dengan input yang diberikan selama proses pengujian adalah data penting yang digunakan untuk memperbaiki kecacatan yang ditemui. Hal ini dilakukan agar dapat mencapai tujuan pengujian yang menunjukkan bahwa *software* yang akan dirilis sesuai dengan kebutuhan pengguna *software* dan sekaligus menjamin kualitas dari *software* yang akan dirilis.

## 5. KESIMPULAN

Pengujian merupakan tahap penting dari proses pengembangan suatu *software*. Investasi pada pengujian *software* membawa dampak besar terhadap kualitas *software*. Untuk memaksimalkan proses pengujian ketika dihadapkan pada keterbatasan waktu dapat dilakukan kolaborasi antara teknik pengujian *fault-based* dan fungsional. Dengan mengimplementasikan strategi berikut:

- Menggunakan kasus uji yang telah digunakan sebelumnya.
- Menentukan prioritas dari deretan kasus uji yang ada berdasarkan banyaknya perkiraan kecacatan terbanyakyang diperoleh.
- Menggunakan pendekatan pencegahan cacat
- Menggunakan produk analisis statis
- Mengunakan pengujian *black – box* sebagai *acceptance test*

**DAFTAR PUSTAKA**

Agarwal, B. B., Tayal, S. P., and Gupta, N., 2010, *Software* Engineering dan Testing, Jones and Bartlett Publishers, Massachusetts

Briski, K. A., et al., 2008, Minimizing Code Defects to Improve *software* Quality and Lower Development Costs, Development Solution White Paper IBM

Lau, M. F., et al., 2007, On Detecting Double Literal Faults in Boolean Expressions, *Springer-Verlag Berlin Heidelberg* LNCS, 4498,pp 55-68

Farooq, S. U dan Quadri, S. M. K, 2010, *Software* Testing – Goals, Principles, and Limitations, *International Journal of Computer Applications* 6(9), pp 0975 – 8887

Kavitha, R., dan Sureshkumar, N., 2011, Fault Based Test Case Prioritization using Integer Linear Programming. European Journal of Scientific Research,57(1), pp 47-52

Kumar, N., dan Rana, A., 2011, A Realistic Approach: RTST to Reduce Cost & Time, International Journal on Computer Science and Engineering (IJCSE), 3(6), pp 2233-2239

Nair, T. R. G., dan Suma, V., 2008, Effective Defect Prevention Approach in *Software* Process for Achieving Better Quality Levels, World Academy of Science, Engineering and Technology, 42,pp 258-262

Patrícia Machado, Auri Vincenzi, and José Carlos Maldonado, 2010, *Software* Testing Overview, Springer-Verlag Berlin Heidelberg 2010, LNCS 6153, pp 1–17

Sibgatullina, T., 2011, Classification And Calculation of Different Types of Cost of *Software* Quality Testing, Annals & Proceedings of DAAAM International 2011, 22(1), pp 1565-1566

Sommerville, I., 2009, *Software* Engineering, Pearson Education, Inc.: Massachusetts

Tian, J., 2005, *Software* Quality Engineering, John Wiley & Sons, Inc., New Jersey

Vincenzi, A., 2010, Functional, Control and Data Flow, and Mutation Testing: Theory and Practice, *Springer-Verlag Berlin Heidelberg* LNCS, pp 6153, 18-58