# Pengujian JPEG 2000 Encoder/Decoder dengan Cara Mutasi

## (*Mutation Testing on JPEG 2000 Encoder/Decoder*)

**Yustina Sri Suharini[1]***

[1]Institut Teknologi Indonesia
Jl. Puspiptek Raya, Serpong
Tangerang Selatan, Banten 15320

**(**Diterima: 14 Oktober 2012; Disetujui: 28 Januari 2013)

**Abstrak**

*Pengujian perangkat lunak merupakan aktivitas yang dilakukan untuk menambah kepercayaan pengembang dan pengguna terhadap perangkat lunak yang diuji. Salah satu cara menguji adalah dengan memberikan sederetan kasus uji sebagai masukan program, lalu mengamati apakah program berperilaku seperti yang telah diprediksikan. Terdapat banyak kasus di mana keluaran yang diharapkan tidak diketahui di awal, yang dikenal sebagai persoalan oracle. Pengujian dengan cara mutasi merupakan sebuah pendekatan yang dilakukan untuk mengatasi persoalan tersebut. Ide dasarnya adalah jika parameter-parameter dalam kode program diubah, maka perilaku program juga akan berubah. Tulisan ini merupakan hasil penelitian yang bertujuan untuk mengetahui rasio atau derajat kemampuan relasi-relasi metamorfik dalam menemukan kesalahan pada program yang diuji. Penelitian dilakukan dengan metoda eksperimental. Untuk keperluan penelitian, digunakan empat kelompok relasi metamorfik, 1023 gambar sebagai kasus uji, dan tiga puluh mutan untuk setiap relasi. Perangkat lunak yang digunakan sebagai obyek yang diuji adalah JPEG-2000 encoder/decoder buatan Michael D. Adams.*

**Kata Kunci** : pengujian mutasi, persoalan oracle, relasi metamorfik, mutan

**Abstract**

*Software testing is an activity to improve reliability of the software. Testing can be done by giving a set of test cases through the software and then observing the output. In software testing terminology, an oracle is a mechanism to decide if the output of the target code is correct given any possible input. Unfortunately, there are many cases in which no oracle is available, that is called as oracle problem. Mutation testing is one of the ways to approach the oracle problem. The idea is simple. If the code is changed, the outcome should be different. This paper is an experimental report in using mutation testing to test Michael D. Adams' JPEG 2000 encoder/decoder. The experiment was done by including four groups of metamorphic relations, 1023 images as test cases, and thirty mutants for every metamorphic relation. The experiment showed variety of the fault detection capability ratio for each metamorphic relation, besides experiences weakness of mutation testing. The both can be used as considerations by tester before using mutation testing*.

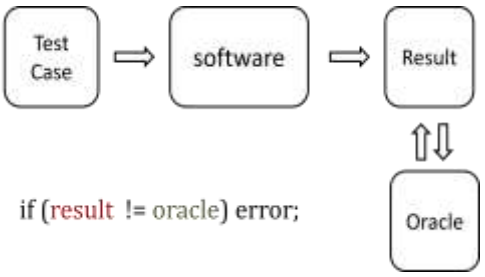
**Keyword** : oracle problem, mutation testing, metamorphic relation, mutant

*Penulis Korespondensi. Telp: +62 21 7561095; fax: +62 21 7560542
Alamat E-mail : yust235@yahoo.com

### 1. Pendahuluan

Pengujian (*testing*) merupakan salah satu aktivitas utama pada siklus hidup pengembangan perangkat lunak (*Software Development Life Cycle* atau SDLC). Pengujian dilakukan untuk meningkatkan kepercayaan pengembang dan pengguna terhadap perangkat lunak yang diuji. Pengujian tidak bisa menemukan keseluruhan *error* atau *defect* yang ada pada perangkat lunak. Hal yang mungkin dilakukan adalah melakukan sebuah pendekatan, yaitu menguji apakah perangkat lunak berperilaku sesuai dengan kebutuhan yang telah disepakati, atau menguji *correctness* perangkat lunak. Pengujian macam ini dilakukan dengan memberi serangkaian kasus uji (*test cases*) sebagai inputan bagi perangkat lunak yang diuji, lalu mengamati apakah perilaku perangkat lunak tersebut sesuai dengan prediksi. Mekanisme untuk memutuskan apakah output sudah betul atas kasus uji yang diberikan, disebut

*oracle*. Misalkan perangkat lunak mempunyai spesifikasi mampu mencetak string sejumlah integer positif yang dimasukkan oleh user, maka oracle yang dapat digunakan antara lain: (1) jumlah string yang dicetak harus sesuai dengan integer positif yang diberikan (2) jika user memberi input selain integer positif, string tidak dicetak.




**Gambar 1.** Pengujian Perangkat Lunak Menggunakan Oracle

Oracle mempunyai peran penting dalam pengujian perangkat lunak. Namun tidak semua perangkat lunak mempunyai oracle, misalnya compiler, cryptosystem, pengenalan pola, serta perangkat lunak dengan nilai pendekatan. Hal ini disebut sebagai persoalan oracle [8].

Salah satu pendekatan yang dilakukan untuk meringankan persoalan oracle adalah dengan melakukan pengujian secara mutasi [7]. Intinya, kode program (*source code*) diubah lalu diamati perilakunya. Ide dasar pengujian mutasi adalah bahwa pengubahan kode program akan mengubah perilaku perangkat lunak. Jika perilaku perangkat lunak masih sama dengan perilaku sebelum diubah, berarti ada sesuatu yang salah dengan program tersebut. Kesalahan bisa berasal dari perangkat lunak, bisa juga berasal dari kasus-kasus uji yang digunakan. *Mutan* adalah sebutan untuk kode yang sudah diubah. Agar pengujian mutasi memenuhi cakupan uji (*coverage*), disiapkan banyak mutan dan banyak kelompok kasus uji untuk tiap mutan. Kasus uji yang ideal adalah kasus uji yang dapat mengenali perubahan, atau yang dikenal sebagai "pembunuh mutan".

JPEG 2000 encoder/decoder merupakan sebuah perangkat lunak yang digunakan untuk mengubah format image dari dan/atau ke bentuk format image yang lain. Hasil encode/decode itu sendiri dapat dikategorikan sebagai sesuatu yang tidak diketahui dengan tepat di awal proses. Dengan kata lain, untuk mendapatkan perkiraan hasil encode/decode diperlukan usaha yang besar dan hasilnyapun bukan sesuatu yang sangat pasti baik dari segi ukuran maupun bentuknya.

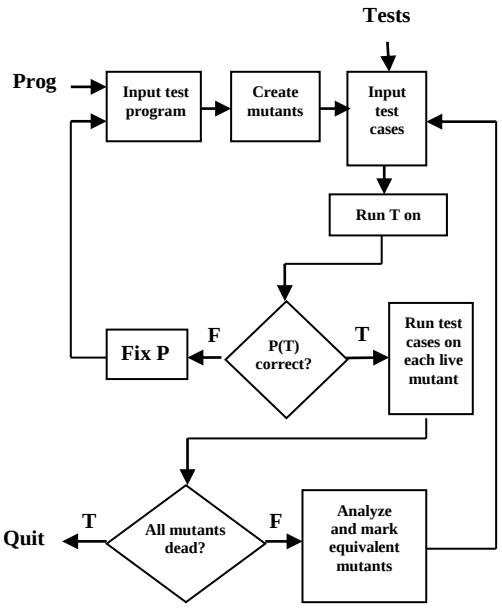
Berdasarkan latar belakang tersebut, dilakukan penelitian untuk mengetahui apakah pengujian mutasi dapat dilakukan terhadap JPEG 2000 encoder/decoder. Tujuan utama penelitian adalah untuk menentukan derajat kemampuan relasi metamorfik dalam mendeteksi kesalahan (*error detection capability ratio*). Obyek yang diuji adalah Jasper-1.900.1, salah satu perangkat lunak dengan fitur utama encode/decode image JPEG 2000 yang dikembangkan oleh Michael D. Adams [1]. Untuk keperluan pengujian, digunakan empat kelompok relasi metamorfik, tiga puluh mutan untuk setiap kelompok, masing-masing diujicobakan pada 1023 image dengan ukuran dan jenis beragam.

## 2. Teori Dasar

Bagian ini berisi teori pengujian mutasi dan relasi metamorfik. Keduanya merupakan teori utama yang mendasari eksperimen.

### Pengujian Mutasi

Mutasi dalam konteks pengujian berarti mengubah kode program asli (*source code*) ke dalam banyak versi mutan lalu membandingkan output setiap mutan dengan output program asli. Satu mutan disisipi satu perubahan, yang mana perubahan ini seolah-olah sebuah kesalahan (*faulty*) yang bisa saja dibuat oleh pemrogram pada saat menulis kode program. Kasus uji yang sama diberikan kepada program asli dan juga kepada mutan. Mutan dikatakan "terbunuh" jika output program yang sudah diubah berbeda dari output program asli.




**Gambar 2.** Proses Mutasi [8]

Pertama-tama, sistem membentuk versi mutan dari program asli. Sebuah versi mutan yang paling dasar dapat dibentuk dengan salah

satu cara ini: menghapus operator, menambah operator, mengubah operand dengan operand lain yang sesuai sintaks, maupun menghapus sebuah baris program.

Langkah berikutnya, deretan kasus uji diberikan kepada program asli maupun kepada mutan. Jika output mutan berbeda dari output program asli, maka mutan ditandai sebagai mutan yang mati/terbunuh. Mutan yang sudah mati tidak dieksekusi lebih lanjut.

Setelah semua deret kasus uji diujikan, nilai mutasi (*mutation score*) dapat dihitung sebagai perbandingan antara mutan yang terbunuh dan jumlah total mutan yang tidak ekivalen. Mutan yang ekivalen terhadap mutan lain, jika ada, dianggap sebagai satu mutan. Penguji berusaha agar mendapatkan nilai mutasi tertinggi, yaitu 1.00, yang berarti seluruh mutan terdeteksi.

**Relasi Metamorfik**

Relasi metamorfik merupakan relasi di antara sederetan kasus uji (input) dan sederetan hasil uji (output) dari eksekusi berulang (*multiple execution*) sebuah perangkat lunak yang diuji [9].

Misalkan [$x_1$, $x_2$, …, $x_k$] adalah sederetan masukan terhadap function f , dengan k ≥ 1, dan misalkan [f ($x_1$), f ($x_2$), …, f ($x_k$)] adalah deretan output terkait. Misalkan juga [f ($x_{i1}$), f ($x_{i2}$), …, f ($x_{im}$)] adalah subderet dari [f ($x_1$), f ($x_2$), …, f ($x_k$)], bisa berupa deret kosong serta [$x_{k+1}$, $x_{k+2}$, …, $x_n$] adalah deret input lain untuk f , where n ≥ k + 1, sehingga [f ($x_{k+1}$), f ($x_{k+2}$), …, f ($x_n$)] merupakan deret output terkait. Lebih lanjut, misalkan terdapat relasi di antara item-item tersebut yaitu $R_i$ ( $x_1$, $x_2$, …, $x_k$, f ($x_{i1}$), f ($x_{i2}$), …, f ($x_{im}$), $x_{k+1}$, $x_{k+2}$, …, $x_n$ ) dan $R_o$ ( $x_1$, $x_2$, …, $x_n$, f ($x_1$), f ($x_2$), …, f ($x_n$) ) sedemikian sehingga $R_o$ selalu betul jika $R_i$ dapat terpenuhi. Di sini $R_i$ dan $R_o$ berupa relasi dari parameter-parameter yang disebutkan sebelumnya. Relasi metamorfik merupakan subset dari produk Cartesian yang dapat dituliskan dengan notasi

MR = { [ $x_1$, $x_2$, …, $x_n$, f ($x_1$), f ($x_2$), …, f ($x_n$) ] | $R_i$ ($x_1$, $x_2$, …, $x_k$, f ($x_{i1}$), f ($x_{i2}$), …, f ($x_{im}$), $x_{k+1}$, $x_{k+2}$, …, $x_n$) → $R_o$ ($x_1$, $x_2$, …, $x_n$, f ($x_1$), f ($x_2$), …, f ($x_n$)) }

...(1)

pada mana jika tidak terjadi ambiguitas maka

MR = jika $R_i$ ($x_1$, $x_2$, …, $x_k$, f ($x_{i1}$), f ($x_{i2}$), …, f ($x_{im}$), $x_{k+1}$, $x_{k+2}$, …, $x_n$) maka $R_o$ ($x_1$, $x_2$, …, $x_n$, f ($x_1$), f ($x_2$), …, f ($x_n$))

....(2)

Selanjutnya [$x_1$, $x_2$, …, $x_k$] disebut kasus uji awal (*initial test cases*) dan [$x_{k+1}$, $x_{k+2}$, …, $x_n$] disebut kasus uji lanjutan (*follow-up test cases*).

Sebagai contoh, sebuah program dengan input berupa koordinat x yang menghitung koordinat y atas garis yang melalui titik ($x_0$,$y_0$). Misal digunakan relasi metamorfik:

$(f(x_1) – y_0) / (x_1 – x_0) = (f(x_2) – y_0)) / (x_2 – x_0)$

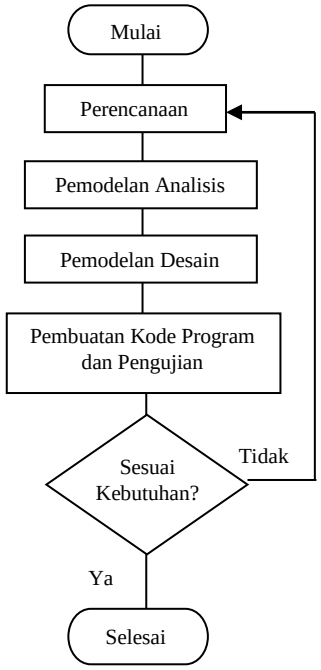
dan nilai titik yang dilewati adalah ($x_0$,$y_0$) = (3,4) serta kasus uji pertama adalah $x_1$=5 memberi output P($x_1$) = 7 yang berarti program yang diuji membaca 5 sebagai input dan memberi output 7. Nilai input lanjutan (follow-up), misal $x_2$=8 seharusnya memberi output P($x_2$) 11.5, maka jika output program ternyata 11, berarti ada sesuatu yang salah dengan program tersebut [10].

**3. Metodologi**

Pengujian mutasi terhadap SUT (Jasper-1.900.1) dilakukan dalam dua tahap, yaitu

(1) Pengembangan pengujian otomatis yang terdiri atas: perencanaan, pemodelan analisis dan desain, konstruksi, integrasi.
(2) Pengujian SUT menggunakan perangkat uji otomatis yang dikembangkan pada tahap pertama, dilanjutkan dengan pengamatan dan analisis hasil uji.

Proses pengembangan perangkat lunak pengujian otomatis ditunjukkan pada Gambar 3.




**Gambar 3.** Tahap Pengembangan Perangkat Lunak Pengujian Otomatis

Pengujian otomatis yang dikembangkan berupa beberapa kode program *shell* dan berbagai mutan. Program yang dibuat digunakan untuk:

[1] menginput setiap kasus uji secara otomatis,
[2] mengeksekusi SUT,
[3] mengeksekusi mutan,
[3] menyimpan setiap hasil uji ke dalam file,

[4] mengatur direktori untuk mengorganisasikan file output
[5] membandingkan setiap output SUT mutan terhadap output SUT asli.

Relasi metamorfik disimbolkan dengan MR. Mengacu pada penelitian pendahuluan oleh Rene Just dkk, penelitian ini menggunakan empat kelompok MR seperti disebutkan berikut ini [6].

$MR_1$:
$R_i$: Tambahkan *offset* ke *color value*
$R_o$: Hanya *DC component* yang terpengaruh
$MR_2$:
$R_i$: Kalikan *color value* dengan koefisien
$R_o$: Tiap *pixel* akan terpengaruh
$MR_3$:
$R_i$: Lakukan *transpose* terhadap matrik *pixel*
$R_o$: *Subband* asli akan di-*transpose*
$MR_4$:
$R_i$: Tambahkan *zero-padding* pada image
$R_o$: *Subband* asli akan digeser

Pada $MR_1$ diujikan beberapa nilai *offset*, demikian juga pada $MR_2$ diujikan beberapa nilai koefisien. Pada $MR_3$ hanya ada satu transpose, sedang pada $MR_4$ diujikan dua macam *padding*, yaitu *left-padding* dan *right-padding*.

## 4. Hasil Penelitian dan Pembahasan

Hasil utama pada ekperimen ini adalah rasio kemampuan pendeteksian kesalahan (*faulty detection capability ratio*) untuk tiap-tiap relasi metamorfik, yang diberikan pada Tabel 1.

**Tabel 1.** Kemampuan Mendeteksi Kesalahan

| Kelompok | Ratio |
|---|---|
| $MR_1$ | 0.13 |
| $MR_2$ | 0.20 |
| $MR_3$ | 1.00 |
| $MR_4$ | 0.80 |

Tabel 1 memberi gambaran bahwa $MR_3$, (dengan $R_i$: Lakukan *transpose* terhadap matrik *pixel* dan $R_o$: *Subband* asli akan di-*transpose* atau disingkat transpose), memberi rasio paling tinggi di antara kelompok MR yang lain, diikuti $MR_4$ (zero padding), $MR_2$ (perkalian koefisien), dan $MR_1$ (penambahan offset). Hal ini menunjukkan bahwa kemampuan tiap relasi metamorfik dalam mendeteksi kesalahan pada program, pada kasus ini, tidak ada yang sama.

Satu siklus eksekusi pengujian otomatis membutuhkan kurun waktu yang panjangnya tergantung ukuran image. Demikian juga tempat penyimpanan hasil uji untuk satu siklus, besarnya sangat bergantung pada ukuran image. Sebagai contoh, 23 image dengan ukuran rata-rata 300KB digunakan sebagai input untuk 5 SUT (yaitu 1 program original, dan 4 program mutan), akan menghasilkan output 1,55GB dalam waktu lebih kurang 70 menit. Sementara itu, 50 image dengan ukuran kira-kira 10-20KB, untuk 5 SUT yang sama akan membutuhkan tempat penyimpanan sekitar 600KB dalam waktu tidak lebih dari 30 menit.

## 5. Kesimpulan

Beberapa kesimpulan yang dapat ditarik dari eksperimen adalah sebagai berikut.

[1] Tingkat kemampuan pendeteksian kesalahan antara relasi metamorfik yang satu terhadap yang lain ternyata berbeda-beda.
[2] Eksperimen yang dilakukan memberi gambaran yang jelas bahwa pengujian dengan cara mutasi "sangat mahal" karena dilakukan secara menyeluruh (*exhaustive*), kurang efisien untuk perangkat lunak berukuran relatif besar (≥50.000 LOC) seperti JasPer-1.900.1.
[3] Pada pengujian mutasi perlu ditambahkan cara lain yang bisa meningkatkan efisiensi pengujian tanpa mengurangi kemampuan penemuan kesalahan.

Pada eksperimen selanjutnya, mutasi dilakukan untuk bagian tertentu dari SUT. Oleh karenanya diperlukan sebuah pendekatan dalam memilih area mutasi (*coverage*), seperti statistik, logika fuzzy, maupun jaringan syaraf tiruan.

## Daftar Simbol

MR = *Metamorphic Relation*
SUT = *Software Under Test*
SDLC = *Software Development Life Cycle*

## Daftar Pustaka

[1] Michael D. Adams. *JasPer-1.900.1*. (Online),(http://www.ece.uvic.ca/~mdadams/jasper/software/jasper-1.900.1.zip, diakses 24 September 2010)

[2] W.K. Chan, T.Y. Chen, Heng Lu. A Metamorphic Approach to Integration Testing of Context-Sensitive Middleware-Based Applications. *Proceedings of the 5th International Conference on Quality Software (QSIC 2005)*, 2005, IEEE Computer Society Press, Los Alamitos, California.

[3] Dorothy Graham, dkk. *Foundations of Software Testing ISTQB Certification.* Thomson, 2007.

[4] Peter Grossman. *Discrete Mathematics for Computing*. 2nd Edition. Australia: MacMillan Education, 1995.

[5] Sarah Heckman. *Mutation Testing*. (Online),(http://openseminar.org/se/modules/165/index/screen.do, diakses 12 Maret 2011)

[6] Rene Just, Franz Schweiggert. "*Automating Software Tests with Partial Oracles in Integrated Environments*", AST '10, May 3-4, 2010, Cape Town, South Africa.

[7] Glenford J. Myers. *The Art of Software Testing*. John Wiley & Sons, 2004.

[8] A. Jefferson Offutt, Roland H. Untch. *Mutation 2000: Uniting the Orthogonal*. (Online),(http://cs.gmu.edu/~offutt/rsrch/papers/mut00.pdf, diakses 14 Februari 2011)

[9] Elaine J. Weyuker. On Testing Non-testable Programs. *The Computer Journal* 1982; 25(4)

[10] Zhang Zhenyu, dkk. An Experimental Study to Compare the Use of Metamorphic Testing and Assertion Checking. *Journal of Software* 2009.

[11] *CU: Defining Unit Tests using Macros*. (Online),(http://www.vietnamesetestingboard.org/zbxe/?document_srl=327240, diakses 24 Mei 2011)