Bug & Debugging #09

AIK21361 (3 sks)

UJI PERANGKAT LUNAK

Nurdin Bahtiar, MT

Materi

1. Apa itu BUG
2. Mengapa BUG Muncul
3. Siklus Hidup BUG
4. Biaya Memperbaiki BUG
5. Kapan Pengujian Dihentikan / Dikurangi
6. Beberapa Istilah
7. Debugging
8. Mengapa Debugging Susah Dilakukan
9. Pertimbangan Psikologis
10. Tiga Kategori Pendekatan Debugging
11. Tiga Pertanyaan Debugging

1. Apa itu BUG

A software bug may be defined as:

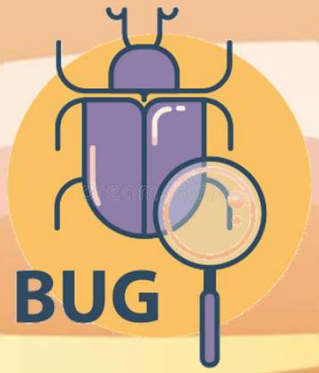
> ***a coding error that causes an unexpected defect, fault, flaw, or imperfection in a computer program***.
>
> *Kesalahan pengkodean yang menyebabkan defect, kesalahan, kecacatan, atau ketidaksempurnaan yang tidak terduga dalam suatu program komputer.*

In other words, if a program does not perform as intended, it is most likely a **bug**.

2. Mengapa BUG Muncul

Beberapa alasan kemunculan BUG:

1. **Unclear software requirements**
2. Constantly **changing software requirements**
3. Also, **fixing a bug** in one part / component of the software
4. **Designing and re-designing**, UI interfaces, integration of modules, database management all these add to the complexity of the software and the system as a whole
5. Fundamental problems with **software design and architecture**.
6. **Rescheduling of resources**, re-doing or discarding already completed work, changes in hardware / software requirements can affect the software too. Assigning a new developer to the project in midway can cause bugs.
7. Programmers usually tend to **rush as the deadline** approaches closer.
8. **Complexity in keeping track** of all the bugs can again cause bugs by itself.



3. Siklus Hidup BUG

Beberapa fase berbeda dari Bug Life Cycle:

1. **Open**:

   A bug is in Open state when **a tester identifies a problem area**

2. **Accepted:**

   The bug is then **assigned to a developer for a fix**. The developer then accepts if valid.

3. **Not Accepted / Won't fix:**

   If the **developer considers the bug as low level** or **does not accept it as a bug**, thus pushing it into Not Accepted / Won't fix state.

3. Siklus Hidup BUG

4. **Pending**:

   A bug accepted by the developer **may not be fixed immediately**. In such cases, it can be put under Pending state.

5. **Fixed**:

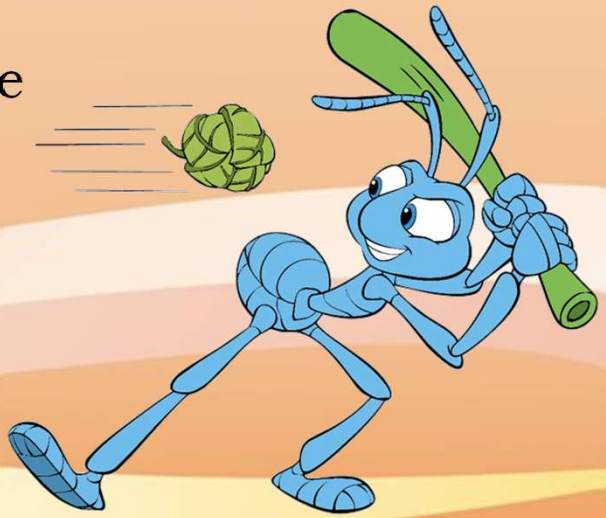
   Programmer will **fix the bug** and **resolves it as Fixed**.

6. **Close:**

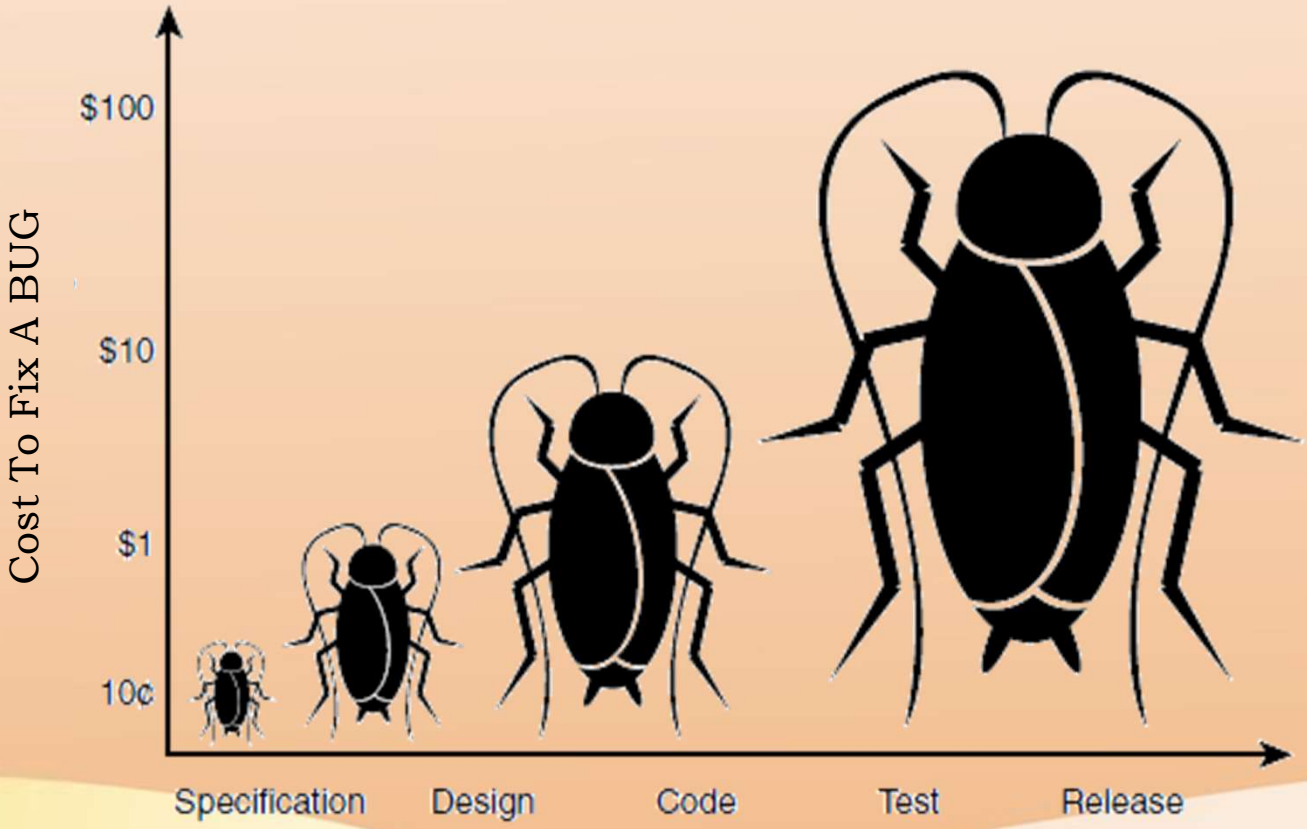
   The fixed bug will be assigned to the tester who will **put it in the Close state**.

7. **Re-Open:**

   Fixed bugs can be re-opened by the testers in case **the fix produces problems elsewhere**.



4. Biaya Memperbaiki BUG



5. Kapan Pengujian Dihentikan / Dikurangi

Memang tidak mudah untuk menentukan kapan suatu pengujian perangkat lunak dihentikan / dikurangi. Berikut beberapa faktor yang dapat membantu memutuskan kapan menghentikan / mengurangi pengujian:

1. **Deadlines** (release deadlines, testing deadlines, etc.)
2. **Test cases completed** with certain percentage passed
3. **Test budget depleted**
4. **Coverage of code** / functionality / requirements reaches a specified point
5. **Bug rate** falls below a certain level
6. Beta or alpha **testing period ends**.

6. Beberapa Istilah

- **Bug**: A coding error that causes an unexpected defect, fault or flaw. In other words, if a program does not perform as intended, it is most likely a bug.
- **Error**: A mismatch between the program and its specification is an error in the program.
- **Defect**: Defect is the variance from a desired product attribute (it can be a wrong, missing or extra data). It can be of two types – Defect from the product or a variance from customer / user expectations. It is a flaw in the software system and has no impact until it affects the user / customer and operational system. 90% of all the defects can be caused by process problems.
- **Failure**: A defect that causes an error in operation or negatively impacts a user/ customer.

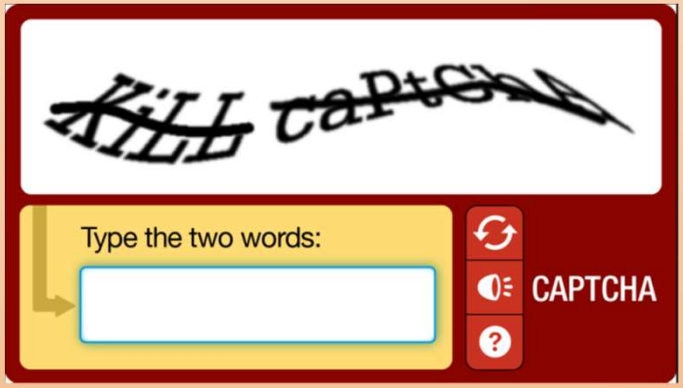
6. Beberapa Istilah

- **Quality Assurance**: Is ***oriented towards preventing defects***. Quality Assurance ensures all parties concerned with the project adhere to the process and procedures, standards and templates and test readiness reviews.
- **Quality Control**: quality control or quality engineering is a ***set of measures taken*** to ensure that defective products or services are not produced, and that the design meets performance requirements.
- **Verification**: Verification ensures the product is designed to deliver all functionality to the customer;
- **Validation**: Validation ensures that functionality, as defined in requirements, is the intended behavior of the product; validation typically involves actual testing and takes place after verifications are completed.

6. Beberapa Istilah

- **Verification**
  → untuk memastikan apakah suatu tahapan / proses sudah dilakukan.

- **Validation**
  → untuk memastikan apakah suatu proses dilakukan dengan benar.



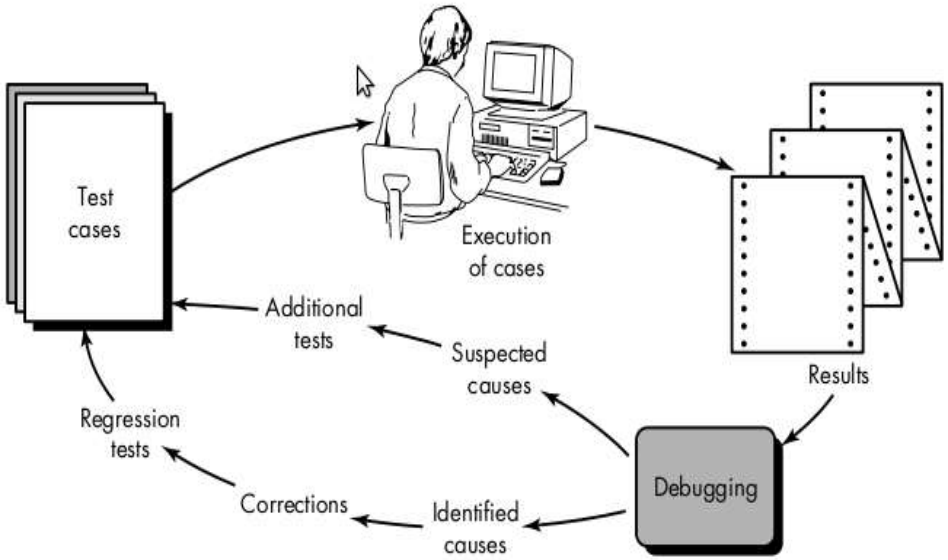
Contoh:
Verification → captcha diisi
Validation → captcha diisi dengan benar

CAPTCHA (*Completely Automated Public Turing Test to Tell Computers and Humans Apart*).

7. Debugging

- *Debugging* terjadi sebagai akibat dari pengujian yang berhasil.
- *Debugging* bukan merupakan pengujian.
- Jika *test case* mengungkap kesalahan, maka *debugging* adalah proses yang menghasilkan penghilangan kesalahan.
- Ada yang mengatakan lain, bahwa *debugging* merupakan proses mental yang dipahami secara buruk yang menghubungkan sebuah *simptom* dengan suatu penyebab.

FIGURE 18.8
The debugging process



8. Mengapa Debugging Susah Dilakukan

Beberapa karakteristik BUG di antaranya:

1. Gejala dan penyebab dapat **jauh secara geografis**
2. Gejala dapat **kadang-kadang hilang** ketika kesalahan lain dibetulkan
3. Gejala dapat **disebabkan oleh sesuatu yang tidak salah** (misalnya pembulatan yang tidak akurat)
4. Gejala dapat **disebabkan oleh kesalahan manusia** yang tidak mudah ditelusuri
5. Gejala dapat merupakan hasil dari **masalah timing**, bukan dari masalah pemrosesan
6. Mungkin **sulit untuk mereproduksi kondisi input** secara akurat (misal: aplikasi real time dimana pengurutan input tidak ditemukan)
7. Gejala dapat **muncul sebentar-sebentar**
8. Gejala dapat **berhubungan dengan penyebab** yang didistribusikan melewati sejumlah tugas yang bekerja pada prosesor yang berbeda.

9. Pertimbangan Psikologis

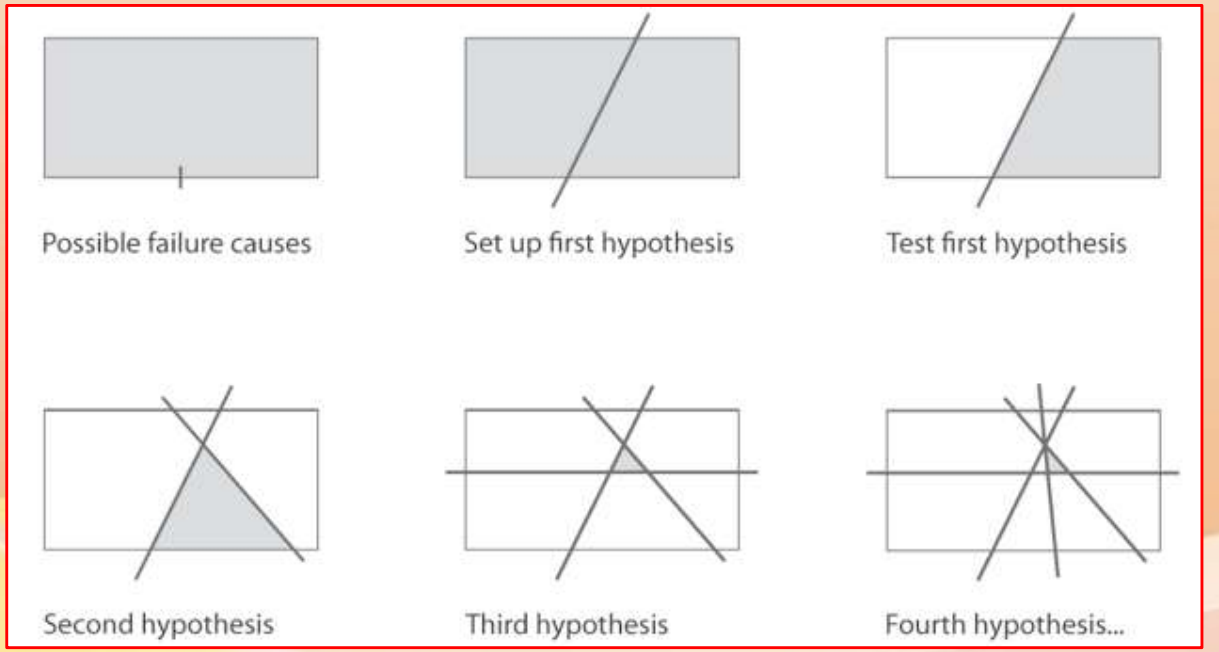
Menanggapi aspek manusia dari *debugging*, Shneiderman menyatakan:

- Debugging merupakan salah satu dari bagian pemrograman yang membuat lebih frustasi.
- Debugging memiliki elemen pemecahan masalah atau pengganggu otak, yang bersama dengan penghindaran kesadaran bahwa Anda melakukan suatu kesalahan.
- Terdapat kekhawatiran yang meningkat dan keengganan untuk menerima kesalahan akan meningkatkan kesulitan tugas.
- Ada keluhan yang sangat mendalam mengenai pembebasan dan pengurangan ketegangan hingga pada akhirnya bug... dikoreksi.

10. Tiga Kategori Pendekatan Debugging

1. Gaya kasar (*brute force*)

Menggunakan filosofi: “Biarkan komputer menemukan kesalahan”, tempat sampah memori dipakai, penelusuran runtime dilakukan, program dibebani dengan statement WRITE. (Kadang disebut juga dengan istilah *Delta Debugging*)



10. Tiga Kategori Pendekatan Debugging

2. Eliminasi penyebab (*cause elimination*)

Data yang berhubungan dengan kejadian kesalahan dikumpulkan.

Hipotesis penyebab dibuat, data digunakan untuk membuktikan hipotesis diterima atau ditolak.

Penjelasan lainnya, yaitu dengan cara:

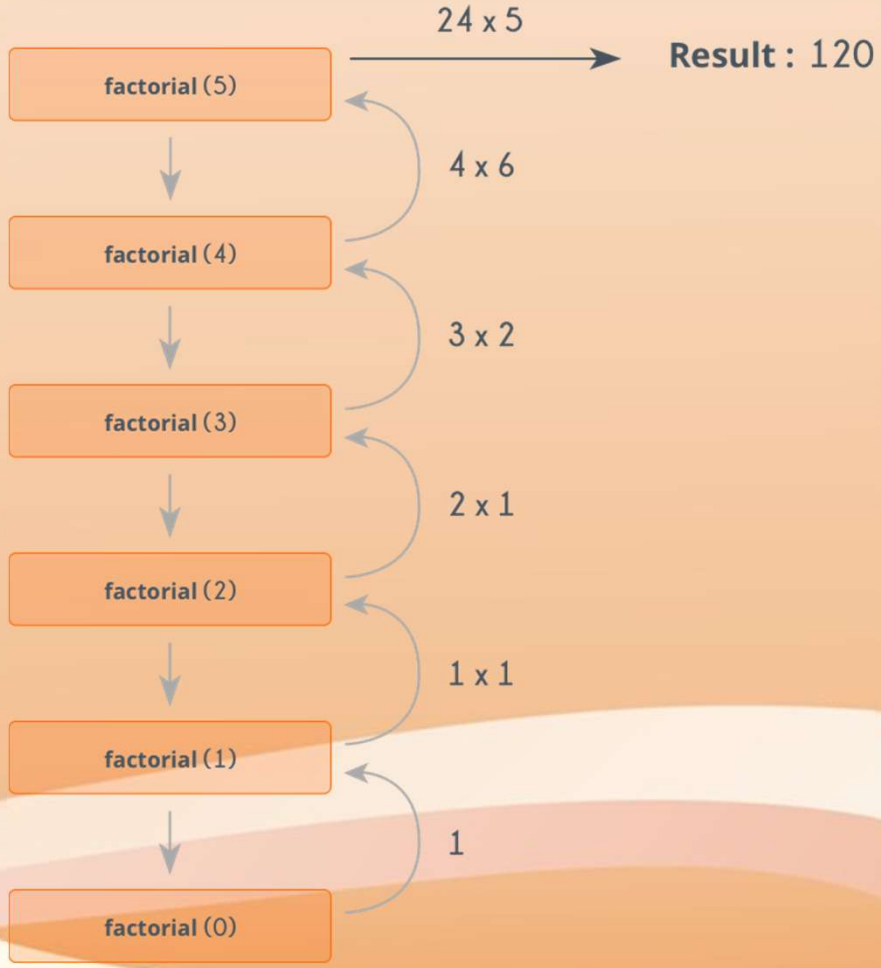
- ✓ Mengumpulkan berbagai penyebab yang mungkin
- ✓ Lalu dilakukan test case untuk mengeliminasi penyebab-penyebab yang tidak relevan.
- ✓ Setelah mengerucut pada satu penyebab yang mungkin, lalu dilakukan penghilangan bug di sekitar area tersebut.

10. Tiga Kategori Pendekatan Debugging

3. Penelusuran balik (*backtracking*)

Secara umum sukses untuk program kecil.

Mulai pada sisi gejala diungkap, kode sumber ditelusuri balik (secara manual) sampai penyebab ditemukan.

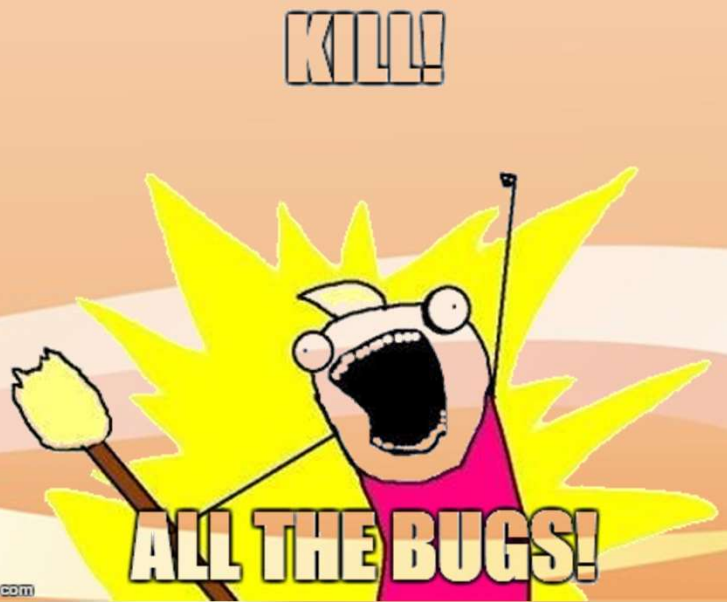


Backtracking Illustration

11. Tiga pertanyaan Debugging

Tiga pertanyaan sederhana yang harus diajukan kepada perekayasa perangkat lunak sebelum melakukan koreksi yang menghilangkan penyebab suatu bug (Van Vleck, 1989):

1. Apakah penyebab bug direproduksi oleh bagian lain dalam program?
2. Apa “bug selanjutnya” yang akan dimunculkan oleh perbaikan yang akan dibuat?
3. Apa yang dapat dilakukan untuk mencegah bug seperti ini?



End of File

Kerjaan

1. Find the bug!

```
for (i=0; i<numrows; i++)
    for (j=0; j<numcols; j++);
          pixels++;
```

Example 1. TYPE: Accidental

Commentary:

Caused by a stray ";" on line 2. Accidental bugs are often caused by stray characters, etc.

Kerjaan

2. Find the bug!

```
int minval(int A, int n) {
     int currmin;
     for (int i=0; i<n; i++)
          if (A < currmin)
          currmin = A;
     return currmin;
}
```

Example 2. TYPE: Missing or improper initialization

Commentary:

Since currmin was never initialized, it could easily start out as the minimum value. Some compilers spot no-initialization errors.

Kerjaan

3. Find the bug!

```
int minval(int A, int n) {
     int currmin=MAXINT;
     for (int i=0; i<n; i++)
          if (A > currmin)
          currmin = A;
     return currmin;
}
```

Example 3. TYPE: Dyslexic

Commentary:

Here, the ">" on line 4 should be "<". Even people who are not normally dyslexic are subject to these types of errors.

Kerjaan

4. Find the bug!

```
switch (i) {
     case 1: do_something(1); break;
     case 2: do_something(2); break;
     case 3: do_something(1); break;
     case 4: do_something(4); break;
     default: break;
}
```

Example 4. TYPE: Mis-copy bug

Commentary:

The cases were generated by copying case 1. Under case 3, the values were not changed as appropriate for the case. Code reuse is good -- but this form of code copying has its dangers!

Kerjaan

5. Find the bug!

```
if (foo = 5)
     foo == 7;
```

Example 5. TYPE: Accidental

Commentary:

Two bugs in one. These are usually caused by accident rather than misunderstanding. The "=" of line 1 should probably be "==" (this one will always evaluate to true), while the "==" of line 2 should almost certainly be "=" (it has no effect).

Kerjaan

6. Find the bug!

```
int i = 5; int j;
int foo(int j) {
      for (i=0; i<j; i++)
          do_nothing();
      return j;
}

void ineedj(void) {
cout << "j is " << j << "\n";
}

main() {
      int j;
      j = foo(i);
      ineedj();
}
```

Example 6.

TYPE: Abused global

Commentary:

This illustrates some fun with global/local variables.

In function foo, j is local and i is global. Since i is being used as a loop variable, this is almost certainly wrong.

Kerjaan

7. Lanjut?

Kunjungi:

http://courses.cs.vt.edu/~cs1206/Fall00/bugs_CAS.html