Applying software validation techniques to Bioperl

Peter van Heusden

http://www.egenetics.com

Agenda

- Reasons to worry about software failure.
- Software validation – defined.
- Applying validation to open source.
- Examples from Bioperl.
- What's next.

Can you trust your software?

- Bioinformatics relies on off-the-shelf software (including open source).
- NIST estimates that poor quality software cost $60 billion in 2002.

Do you know what

you are putting in your pipeline????

Cost of software failure

- Drug discovery pipeline cost about $900 million per drug, over 10+ years.
- Incorrect results = lost time = lost money (not to mention lost opportunities!)
- Worst case scenario: late failure
  - Mars Climate Orbiter = $653 million

Cost of clinical software failure

- In clinical applications, software failure can mean lost lives
  - Therac-25
- Regulatory agencies (e.g. FDA) are promoting software validation
  - 7.7% of device recalls due to software failure
- Pharmacogenomics impact
  - Research and clinical applications merging

So what is software validation?

- Providing a guarantee that software works as advertised
- FDA definition:

"confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that particular requirements implemented through software can be consistently fullfilled"

How software is validated

- What is involved in validation:
  - Documentation
  - Manual examination (“eyeballing”) and testing
- In commercial environments, typically through formal procedures (akin to TQM and ISO 9001)
  - Focused on providing an “audit trail”
- Validation is ongoing
  - 80% of software defects introduced when software changed after initial production

Open source is “scary”

- “Permanent beta” status.
- Open source = collaborative development.
  - Many people over many different enterprises.
  - Many different user requirements.
- Formal procedures are impossible to apply.

Many failure points increase risk

- Many potential “breaking points”
  - Upgrade in-house application
  - Upgrade Bioperl
  - Upgrade Perl
  - Upgrade operating system
- Failure in any one could “break” the system and cause expensive down-time.
  - Minimize breakdowns.
  - If failure occurs, minimize time to identify and resolve.

Evaluation: Bioperl

- “Many eyes make all bugs shallow” approach.
- Unfortunately, not all components are used equally frequently.

Bioperl status quo

- Bug reports from users are supplemented with automated regression testing.
- Regression testing is poor at catching ‘semantic errors’:
  - “True but useless” results (e.g. in SeqIO).
  - Misleading exception messages.

What EG does for Bioperl

Take on the ‘drudgework’ of validating the Bioperl code base.

1. Write documentation that clearly explains the use of the existing code (the “contract”).
2. Examine existing code to check semantics.
3. Write test suite to “test the contract”.
4. Final output: a safe, “off the shelf” Bioperl.

Example: SeqIO

- Core part of Bioperl, widely used.
- Constructor semantics try and mirror Perl file handling semantics.

But see what it does….

```
use IO::File;
use Bio::SeqIO;

my $fh = IO::File->new("/tmp/fastaa.txt");
my $input = Bio::SeqIO->new(-fh => $fh);
```

```
------------ EXCEPTION  ------------
MSG: Unknown format given or could not determine it []
STACK Bio::SeqIO::new /usr/lib/perl5/site_perl/5.8.2/Bio/SeqIO.pm:378
STACK toplevel break_SeqIO_3.pl:8
-------------------------------------
```

```
use IO::File;
use Bio::SeqIO;
use Bio::PrimarySeq;

my $fh = IO::File->new("/tmp/fasta.txt");
my $out_fh = IO::File->new("/tmp/newfasta.txt");

my $input = Bio::SeqIO->new(-fh => $fh);
my $output = Bio::SeqIO->new(-fh => $out_fh, -format => 'fasta');

my $seq = new Bio::PrimarySeq(-id => 'ACC1', -seq => 'ACGGGAAAC');
$output->write_seq($seq);
```

```
>ACC1
ACGGGAAAC
```

```
use IO::File;

use Bio::SeqIO;

my $cfg_file = "break_SeqIO.txt";

my $reference_seq_file;

%params = read_config($cfg_file);
$reference_seq_file = $params{'ReferenceSeq'};
my $input = Bio::SeqIO->new(-file => $reference_seq_file);
my $reference_seq = $input->next_seq();
$input->close();

my $comparison_seq_file = $ARGV[0];
$input = Bio::SeqIO->new(-file => $comparison_seq_file);
my $comparison_seq = $input->next_seq();
$input->close();

print “matched!\n” if ($reference_seq->seq() eq $comparison_seq->seq());
```

Can't call method "seq" on an undefined value at break_SeqIO.pl line 21.

Example: SeqIO (cont)

- Constructor contains number of semantic errors
- Choices:
  - Either document semantics and risk.

    OR

  - Convince community to change semantics.

What is the outcome?

- Contributions to Bioperl
  - Ongoing, systematic testing of Bioperl code.
  - Patches + new code added to the public code base.
- Bioperl Validation and Support Service
  - Testing frameworks.
  - Upgraded documentation.
  - Validation of in-house code that utilizes Bioperl.
  - Customer support.

Users have confidence that modules will work as documented in their particular environment.

Where to from here?

- Validation improves usability ***and*** code.
- Electric Genetics aims to improve “commercial confidence” in Bioperl.
- Increased confidence = increased users.
- Increased users = increased contributors.
- Develop a viable methodology for validating open source software.
  - Applicable to BioJava, etc.

No more Therac-25s!