Beyond Error Handling

Using Design to Prevent Errors

Michael Feathers
R7K Research & Conveyance
IF "Claim Charge Income Account" = '' THEN BEGIN
  ASSERTERROR ERROR('');
  COMMIT;
  IF CONFIRM(MissingData,TRUE,TABLECAPTION,FIELDCAPTION("Claim Charge Income Account")) THEN BEGIN
    PAGE.RUNMODAL(PAGE::"Payment Methods","Payment Method");
    COMMIT;
  END;
END;

IF "Claim Capital Tax Bal. Account" = '' THEN
  IF CONFIRM(MissingData,TRUE,TABLECAPTION,FIELDCAPTION("Claim Capital Tax Bal. Account")) THEN BEGIN
    ASSERTERROR ERROR('');
    COMMIT;
    PAGE.RUNMODAL(PAGE::"Payment Methods","Payment Method");
    COMMIT;
  END;

IF "Claim Charge Income Account" = '' THEN
  ERROR('');

IF "Claim Capital Tax Bal. Account" = '' THEN
  ERROR('');
Why is Error Handling Code So Pervasive?
void Foo()
{
    _logger.Info("Getting first bar from database...");

    Bar bar = dbContext().Bars.First();

    _logger.Info("First bar returned from database, id = {0}", bar.Id);

    _logger.Info("Doing something on bar with id = {0}", bar.Id);
    bool someCondition = bar.DoSomething();

    _logger.Info("Something done on bar with id = {0}, response = {1}", bar.Id, someCondition);

    if (someCondition)
    {
        _logger.Warn("Adding new FooBar to database");
        dbContext().FooBars.Add(new FooBar());
        _logger.Warn("Added new FooBar to database successfully");
    }
}
Logging and Error Handling
GROWING OBJECT-ORIENTED SOFTWARE, GUIDED BY TESTS

Steve Freeman
Nat Pryce
PaymentProcessor
PaymentProcessor

Insufficient Funds
PaymentProcessor

- Insufficient Funds
- Invalid Payment Method
void Foo()
{
    _logger.Info("Getting first bar from database..." redundancy removed);

    Bar bar = dbContext().Bars.First();

    _logger.Info("First bar returned from database, id = {0}", bar.Id); is redundant

    _logger.Info("Doing something on bar with id = {0}", bar.Id); redundancy removed
    bool someCondition = bar.DoSomething();

    _logger.Info("Something done on bar with id = {0}, response = {1}", bar.Id, someCondition); redundancy removed

    if (someCondition)
    {
        _logger.Warn("Adding new FooBar to database");
        dbContext().FooBars.Add(new FooBar());
        _logger.Warn("Added new FooBar to database successfully");
    }
}
Use an exception when you can’t tell in advance whether an operation will succeed or fail.

- Bertrand Meyer
When does this happen?
Our system
Our system
Our system

the creamy centre
Our system

the creamy centre

protective shell
Our system

the creamy centre

protective shell
Our system

- protective shell
- the creamy centre
  
trusted zone
Our system

*the creamy centre*

trusted zone

*(it should just work)*
IO is the biggest unknown - it’s a system edge
Translate from “outside world errors” immediately.
public class Sale {
    private Inventory inventory = new Inventory();
    private ArrayList<Item> items = new ArrayList<>();

    public void addBarcode(String barcode) {
        Item item = inventory.itemForBarcode(barcode);
        items.add(item);
    }
}
public class Sale {
    private Inventory inventory = new Inventory();
    private ArrayList<Item> items = new ArrayList<>();

    public void addBarcode(String barcode) {
        Item item = inventory.itemForBarcode(barcode);
        items.add(item);
    }
}
Where is the outside world here?
private void HandleMessageCompletion(
    Message message,
    TransactionScope tx,
    OpenedQueue messageQueue,
    Exception exception,
    Action<CurrentMessageInformation, Exception> messageCompleted,
    Action<CurrentMessageInformation> beforeTransactionCommit)
{
    var txDisposed = false;
    if (exception == null)
    {
        try
        {
            if (tx != null)
            {
                if (beforeTransactionCommit != null)
                    beforeTransactionCommit(currentMessageInformation);
                tx.Complete();
                tx.Dispose();
                txDisposed = true;
            }
            try
            {
                if (messageCompleted != null)
                    messageCompleted(currentMessageInformation, exception);
            }
            catch (Exception e)
            {
                Trace.TraceError("An error occured when raising the MessageCompleted event, ",
                               "the error will NOT affect the message processing" + e);
            }
        }
        return;
    }
}
Better?

```java
public class Sale {
    private Inventory inventory = new Inventory();
    private ArrayList<Item> items = new ArrayList<Item>();

    public void addBarcode(String barcode) {
        Item item = inventory.itemForBarcode(barcode);
        if (item != null) {
            items.add(item);
        }
    }
}
Better?

```java
public class Sale {
    private Inventory inventory = new Inventory();
    private ArrayList<Item> items = new ArrayList<Item>();

    public void addBarcode(String barcode) {
        Item item = inventory.itemForBarcode(barcode);
        if (item != null) {
            items.add(item);
        }
    }
}
```

Ifs without elses are sometimes like swallowing exceptions
Better?

```java
public class Sale {
    private Inventory inventory = new Inventory();
    private ArrayList<Item> items = new ArrayList<Item>();

    public void addBarcode(String barcode) {
        try {
            Item item = inventory.itemForBarcode(barcode);
            items.add(item);
        } catch (ItemNotFoundException e) {
        }
    }
```
Better-er?

```java
public class Sale {
    private Inventory inventory = new Inventory();
    private ArrayList<Item> items = new ArrayList<Item>();

    public void addBarcode(String barcode) {
        inventory.itemForBarcode(barcode, (Item item) -> items.add(item));
    }
}
```
Our system

the creamy centre

protective shell
Tell, Don't Ask

Alec Sharp, in the recent book *Smalltalk by Example* [SHARP], points up a very valuable lesson in few words:

> Procedural code gets information then makes decisions. Object-oriented code tells objects to do things.

— Alec Sharp
barcode

Inventory

item

Sale
Ultimately, we have to think about what our error means to the business
public class Sale {
    private Inventory inventory = new Inventory();
    private ArrayList<Item> items = new ArrayList<Item>();

    public void addBarcode(String barcode) {
        Item item = inventory.itemForBarcode(barcode);
        if (item != null) {
            items.add(item);
        } else {
            listener.itemNotFound(barcode);
        }
    }
}
It’s all a matter of design..
Fail Fast
Carry On
Ignore
Record
Tended and Untended Systems are Significantly Different
the creamy centre

Passing Through
public class Sale {
    private Inventory inventory = new Inventory();
    private ArrayList<Item> items = new ArrayList<Item>();

    public void addBarcode(String barcode) {
        items.add(inventory.itemForBarcode(barcode));
    }
}
class NotFound extends Item {
    NotFound() {
        super("Item not found", new Money(0));
    }
}
There are many variations of Special Case Object:

- Null Object
- NaN
- Empty List
- Option types (Maybe Monad)
In pipeline programming.

```java
int sum = widgets.stream()
    .filter(b -> b.getColor() == RED)
    .mapToInt(b -> b.getWeight())
    .sum();
```
Special Case Objects are fine when we are careful about exogenous properties
Domain Extension
def span_count ary
    return 0 if ary.size == 0
    count = 0
    if ary[0] > 0
        count = 1
    end
    i = 0
    while i < ary.size - 1 do
        if ary[i] == 0 && ary[i+1] != 0
            count = count + 1
        end
    end
    return count
end
def span_count ary
    ([0] + ary).lazy
        .each_cons(2)
        .count {|c,n| c == 0 && n != 0 }
end
Noticeable Error Handling is a Symptom of Bad Design
Generalize to maintain the Creamy Centre
\texttt{mix = unwords . map unwords . transpose . map (concat . repeat . words)}
What is the square root of -1?
the creamy centre
Avoid Tunneling
The string is a stark data structure and everywhere it is passed there is much duplication of process. It is a perfect vehicle for hiding information.

*Alan Perlis*
Replace the type code with subclasses.
Total Functions

Description

The C library function `double sqrt(double x)` returns the square root of `x`.

Declaration

Following is the declaration for sqrt() function.

```
double sqrt(double x)
```
Total Functions

def temporal_correlation_of_classes(events):
    events.group_by({e: [e.day, e.committer]})
    .values()
    .map(lambda e: e.map(&:class_name).uniq.combination(2).to_a)
    .flatten(1)
    .pairs()
    .freq_by({e: e})
    .sort_by({p: p[1]})
    end
Noticeable Error Handling is a Symptom of Bad Design
Noticeable Error Handling is a Symptom of Bad Design

Consider ALL the cases to make design better
The mechanism you use is not as important as preserving the creamy centre and challenging the error
Creamy Center