W4156

Debugging & Defect Management

Whoops

The code doesn’t do what we want in development

OR

We have a ‘bug report’ from a higher environment.

What now?

Agenda

- ❏ Debugging
- ❏ Approach to Debug Methodically
- ❏ Tool Support
- ❏ Worked Example

Debugging

Debugging

- We know the meaning of ‘bugs’ therefore debugging is obvious

More formally:

- Debugging is the *methodical process* of *locating* and *removing* defects

Trivial Debugging

In the simplest scenario debugging is easy.

1. Code is small
2. Bug occurs is current version of the code
3. Code is deterministic
4. System is observable
5. Easy to replicate issue in development
6. Easy to ‘step through’ program
7. Single root cause
8. Manifestation directly traceable back to root cause

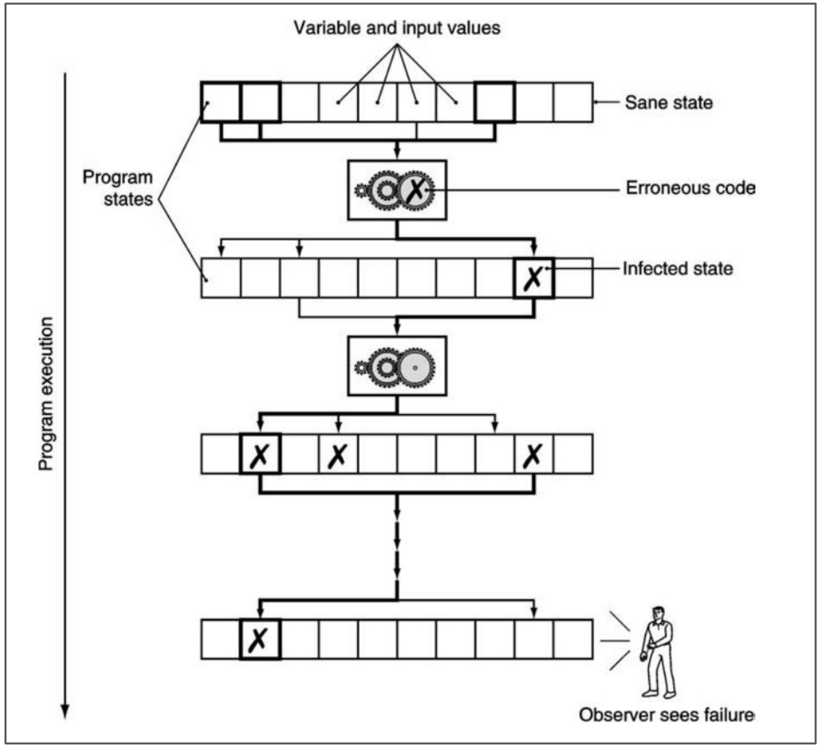
Difficult Debugging

In the simplest scenario debugging is easy.

1. Very very large codebase
2. Older version of the code
3. Vague bug report
4. V. difficult to replicate (don’t know when or why defect triggered)
5. Code is not-deterministic (we may run 10-100 times to replicate bug)
6. Difficult to replicate in development
7. System difficult to observe / complex deployment
8. Very difficult to step through
9. Multiple root causes
10. Series of transformations between manifestation and root cause …

See tales from the front

Bug Trajectory



When Programs FAIL - Zeller

Zeller defines the trajectory as

1. Latent erroneous code
2. *Erroneous code is executed*
3. Defect creates *infection* (program state differs from intended)
4. *Infection propagates* (further state/layers deviate)
5. Cascading
6. Infection finally manifests in a visible failure

Approach

How do we debug non-trivial bugs?

Debugging Structured Approach

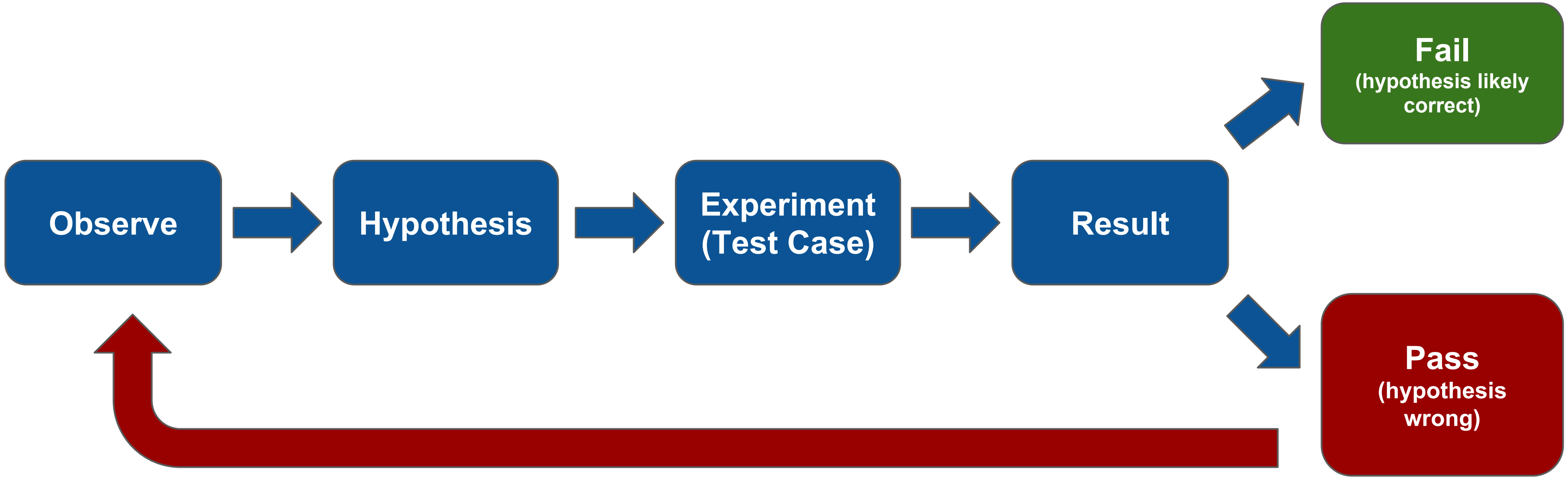
Zeller defines the debugging process as:

Track the problem
Reproduce the failure
Automate the test case (test case *fails* == we have recreated)
Find infection origins
Focus on the most likely origins
Isolate the infection chain
Correct the defect

To which I would add:

Understand *why* the bug occurred
Remediate process/architecture to minimize future recurrence

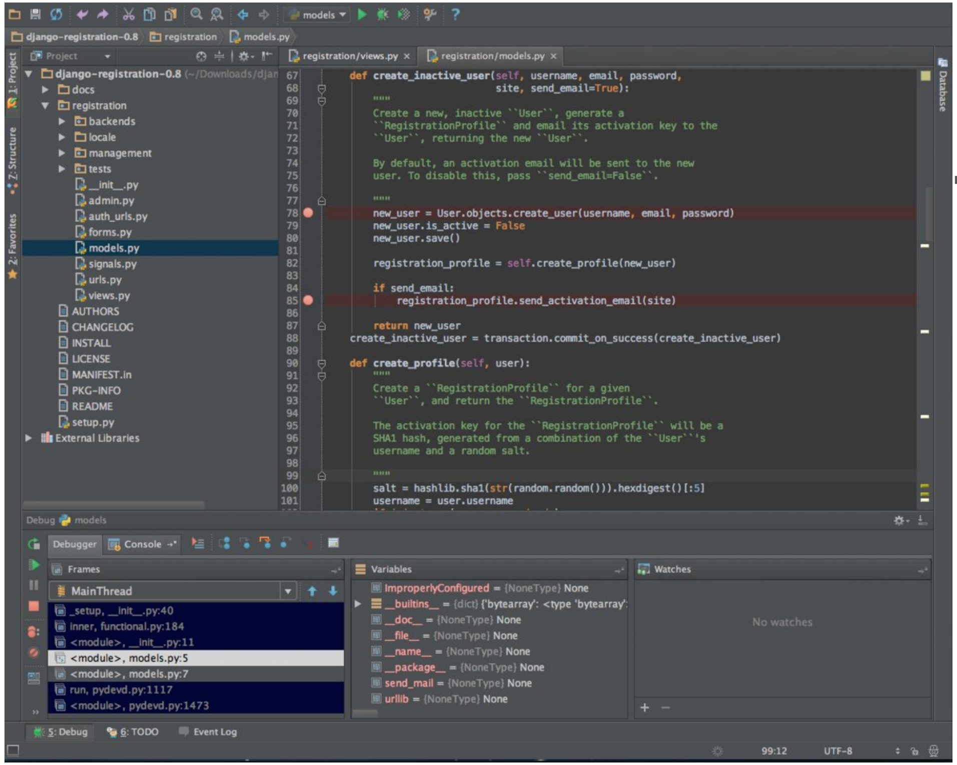
How to Reproduce? Scientific Method



(Remember: we *want* the test case to fail meaning we defined correct/expected behavior and that didnt happen)

Tool Support

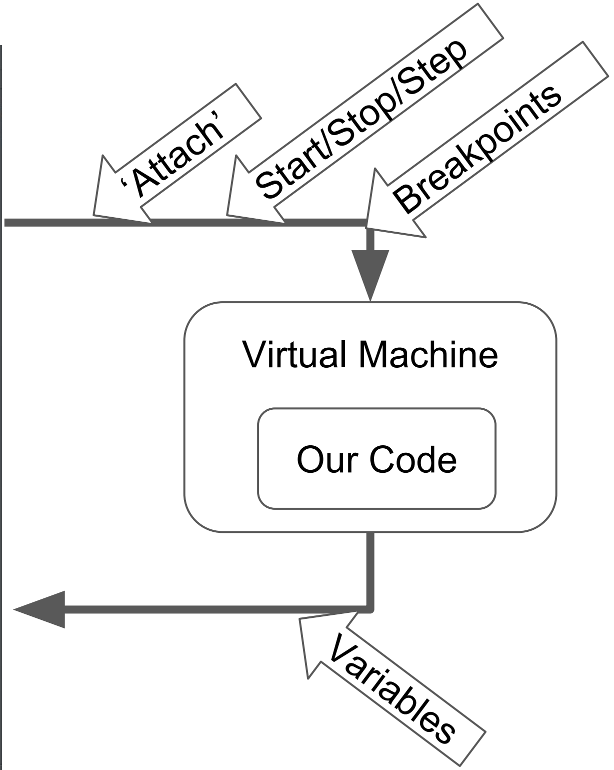
IDE Debugging



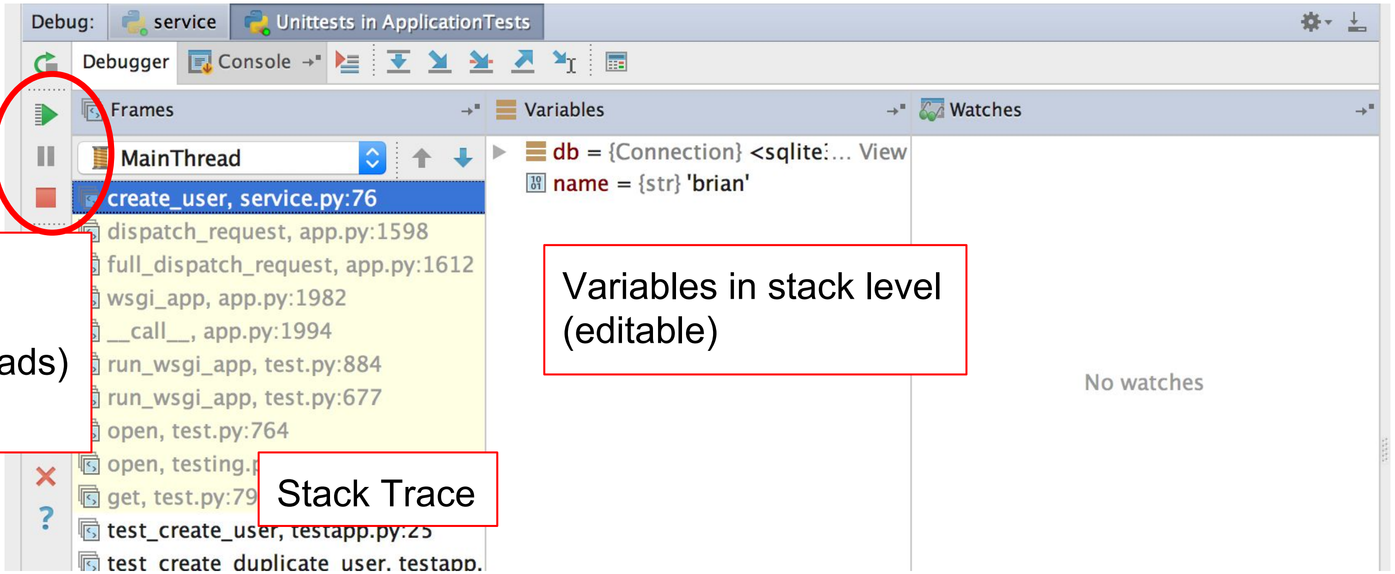
```python
    def create_inactive_user(self, username, email, password,
                             site, send_email=True):
        """
        Create a new, inactive ``User``, generate a
        ``RegistrationProfile`` and email its activation key to the
        ``User``, returning the new ``User``.

        By default, an activation email will be sent to the new
        user. To disable this, pass ``send_email=False``.

        """
        new_user = User.objects.create_user(username, email, password)
        new_user.is_active = False
        new_user.save()

        registration_profile = self.create_profile(new_user)

        if send_email:
            registration_profile.send_activation_email(site)

        return new_user
    create_inactive_user = transaction.commit_on_success(create_inactive_user)

    def create_profile(self, user):
        """
        Create a ``RegistrationProfile`` for a given
        ``User``, and return the ``RegistrationProfile``.

        The activation key for the ``RegistrationProfile`` will be a
        SHA1 hash, generated from a combination of the ``User``'s
        username and a random salt.

        """
        salt = hashlib.sha1(str(random.random())).hexdigest()[:5]
        username = user.username
```

```
MainThread
_setup, __init__.py:40
inner, functional.py:184
<module>, __init__.py:11
<module>, models.py:5
<module>, models.py:7
run, pydevd.py:1117
<module>, pydevd.py:1473
```

```
ImproperlyConfigured = {NoneType} None
__builtins__ = {dict} {'bytearray': <type 'bytearray'
__doc__ = {NoneType} None
__file__ = {NoneType} None
__name__ = {NoneType} None
__package__ = {NoneType} None
send_mail = {NoneType} None
urllib = {NoneType} None
```



IDE Support



```
create_user, service.py:76
dispatch_request, app.py:1598
full_dispatch_request, app.py:1612
wsgi_app, app.py:1982
__call__, app.py:1994
run_wsgi_app, test.py:884
run_wsgi_app, test.py:677
open, test.py:764
open, testing.p
get, test.py:79
test_create_user, testapp.py:25
test_create_duplicate_user, testapp.
```

```
▶ db = {Connection} <sqlite:... View
  name = {str} 'brian'
```

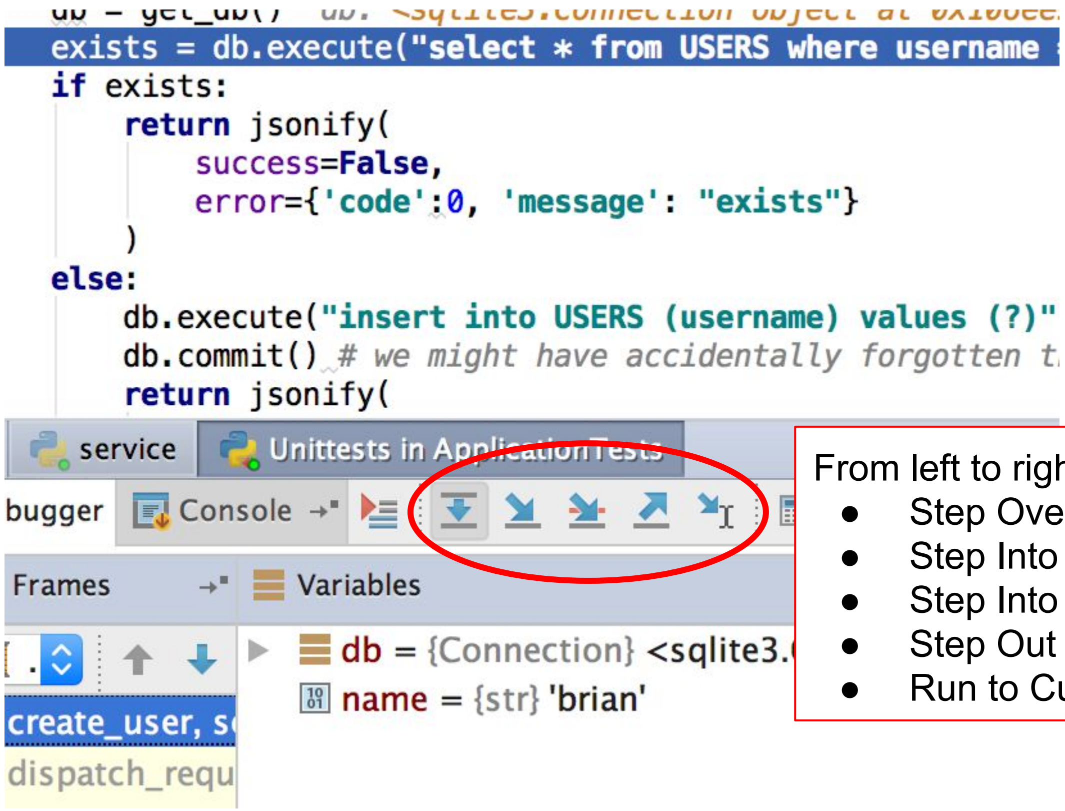
Top to bottom

- Run/Continue
- Pause (all theads)
- Stop/Kill

Variables in stack level (editable)

Stack Trace

IDE Stepping



```
    db = get_db()  db: <sqlite3.Connection object at 0x100ee
    exists = db.execute("select * from USERS where username 
    if exists:
        return jsonify(
            success=False,
            error={'code':0, 'message': "exists"}
        )
    else:
        db.execute("insert into USERS (username) values (?)"
        db.commit() # we might have accidentally forgotten t
        return jsonify(
```

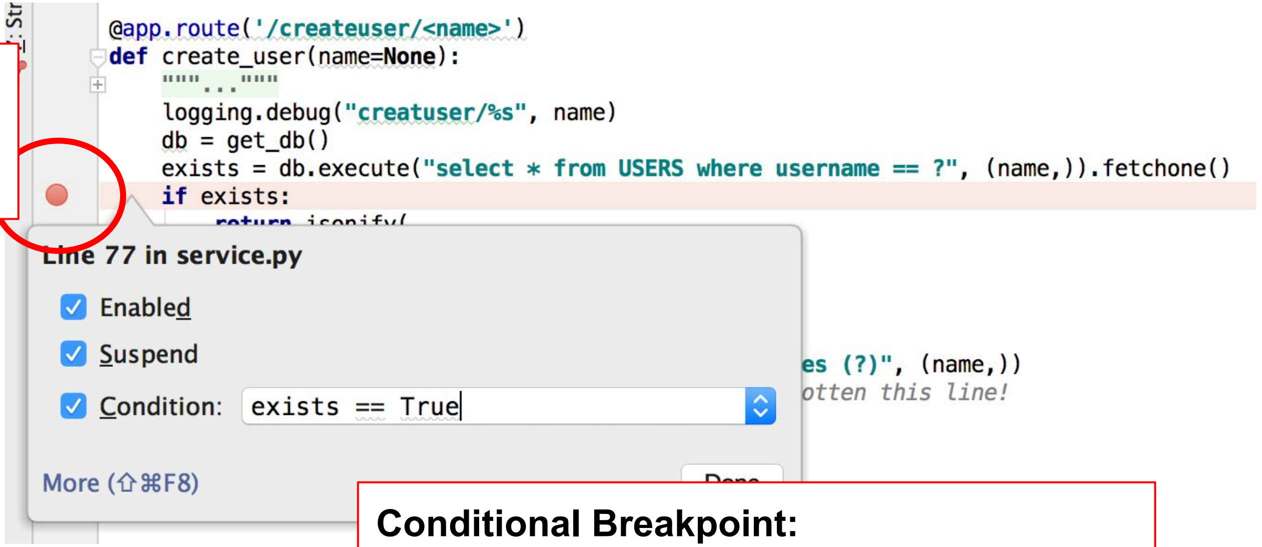
```
db = {Connection} <sqlite3.
name = {str} 'brian'
```

From left to right

- Step Over
- Step Into
- Step Into My Code
- Step Out
- Run to Cursor

IDE Breakpoints

Breakpoint:
Stop always when line hit
(click on line number)



```
@app.route('/createuser/<name>')
def create_user(name=None):
    """..."""
    logging.debug("creatuser/%s", name)
    db = get_db()
    exists = db.execute("select * from USERS where username == ?", (name,)).fetchone()
    if exists:
        return jsonify(
es (?)", (name,))
otten this line!
```

Line 77 in service.py

- [x] Enabled
- [x] Suspend
- [x] Condition: exists == True

Conditional Breakpoint:
Only stop when *condition met*
(Right click on breakpoint to edit condition)

Worked Example

Example

Yo developers of W4156.

Remember that ‘user service’ you created to help add, list, count and delete users?

There seems to be some bug where when I list users it returns 0 users when I know users have been created?

We are losing $5k each hour this bug exists. Can you fix it quickly please!

Let’s apply Process

Track the problem	Entered into bug database
Reproduce the failure	Explore/form hypothesis till recreated
Automate the test case (which will fail)	Wrote a repeatable unit test (test case fails)
Find infection origins	Traced upstream
Focus on the most likely origins	
Isolate the infection chain	Understood
Correct the defect	Fixed (test case now passes)

Does model explain example?

1. Latent erroneous code
2. *Erroneous code is executed*
3. Defect creates *infection* (program state differs from intended)
4. *Infection propagates* (further state/layers deviate)
5. Cascading
6. Infection finally manifests in a visible failure

Bug in the code on db interaction

Returned correct result

But incorrect *state* created

(our example had 1-layer)

Call returned incorrect result

Apply in Project

Dont be ad-hoc stop, add println, start, repeat

1. ***Track defects*** as work to be done (other than ones during dev/non-committed code)
    a. We will use trello for simple task tracking (create a label)
2. ***Fix defects*** **<u>before</u>** working on new features
3. ***Replicate*** the bug in a test rig / write a test that fails (exposes the defect)
4. Hunt for the bug in a ***systematic*** fashion
5. ***Fix*** the bug (and test should pass)
6. Don't beat yourself up about each defect but do think about
    a. *why* this bug was created (gap in requirements, design, coding practices, ..?)
    b. *why* was bug not caught in testing? What type of test would have caught this?

Personal Advice

1. Don’t get frustrated and try random things
2. Don’t println, stop, start (it fills the codebase with print statements, takes too long)
3. Don’t fall into the trap of blaming the compiler (it may be but it is unlikely!)
4. Computers are *relatively* deterministic (if bug appears non-deterministic think *why / sources*)
5. Follow the methodical debugging process
6. Get a rubber duck
7. Go for a walk
8. Its a skill. Over time you will
   a) write fewer defects
   b) write code easier to debug
   c) become more analytical at ‘sherlocking’

Pop Quiz

Question	Answer
Defects are created during the construction phase <True/False>	
The cost of bugs <grows/falls> the later they are discovered in the process	
Bugs that are harder to find may be because there is a gap between the ________ which when executed creates ________ which may then further _______ before eventually manifesting in a ________ (choose from: failure, latent error, cascade, infected state)	
The phases of a structured debugging process are ……?	
When bugs are reported they should first be ________ in a _________	
Generally _______ are higher priority than _________ (choose from: features, bugs)	

Reading

Reading	Optionality
Why programs fail chapter 1 & 6	Required
This or this (much wider but very fun)	Optional

Cost of Bug

Cost of Bug =

Disruption to customer +

Long term loss of customers +

Cost of Operations Time Firefighting +

Cost of Tactical Fix / Workaround +

Cost of Developer Replicating Issue +

Cost of Developing Fix +

Cost of Coordinating deployment of fix +

Cost of Patching other branches +

Cost of Explaining to management +

Cost of Explaining to customers