Andrei Lisnic

Dependency Injection Approaches

Who am I?

Developer for 9+ years
Most of them in Ruby/Ruby on Rails, but did a lot of JavaScript too
Joined Planable in April, started doing TypeScript/React

https://keybase.io/andrei

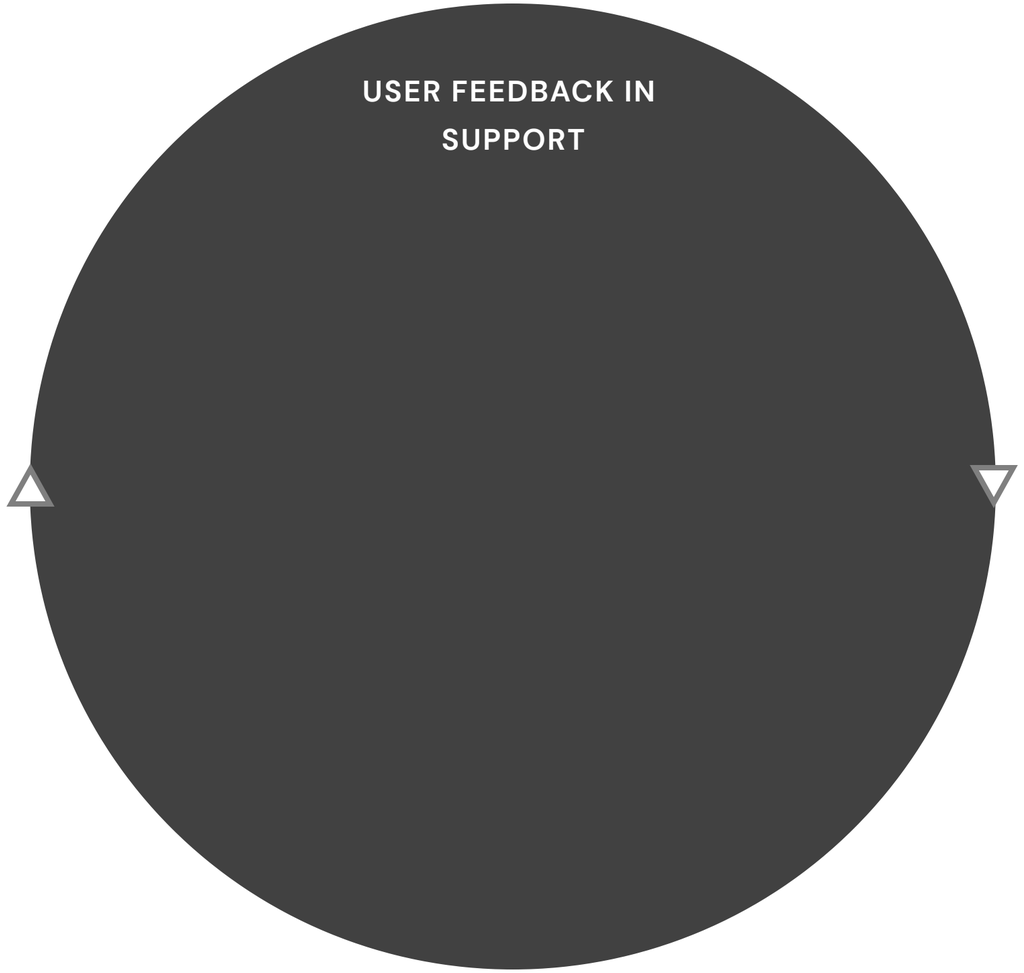
What's a unit anyway?

```
import TwitterClient from 'twitter'
import isPostValid from '~/path/to/isPostValid'

export default function postToTwitter(text) {
  if (!isPostValid({ text }))
    throw new Error('invalid post')

  const payload = makeTwitterPayload(text)
  TwitterClient.post(payload)

  return true
}

function makeTwitterPayload() {
  // ...snip
}
```

Unit – a module export.

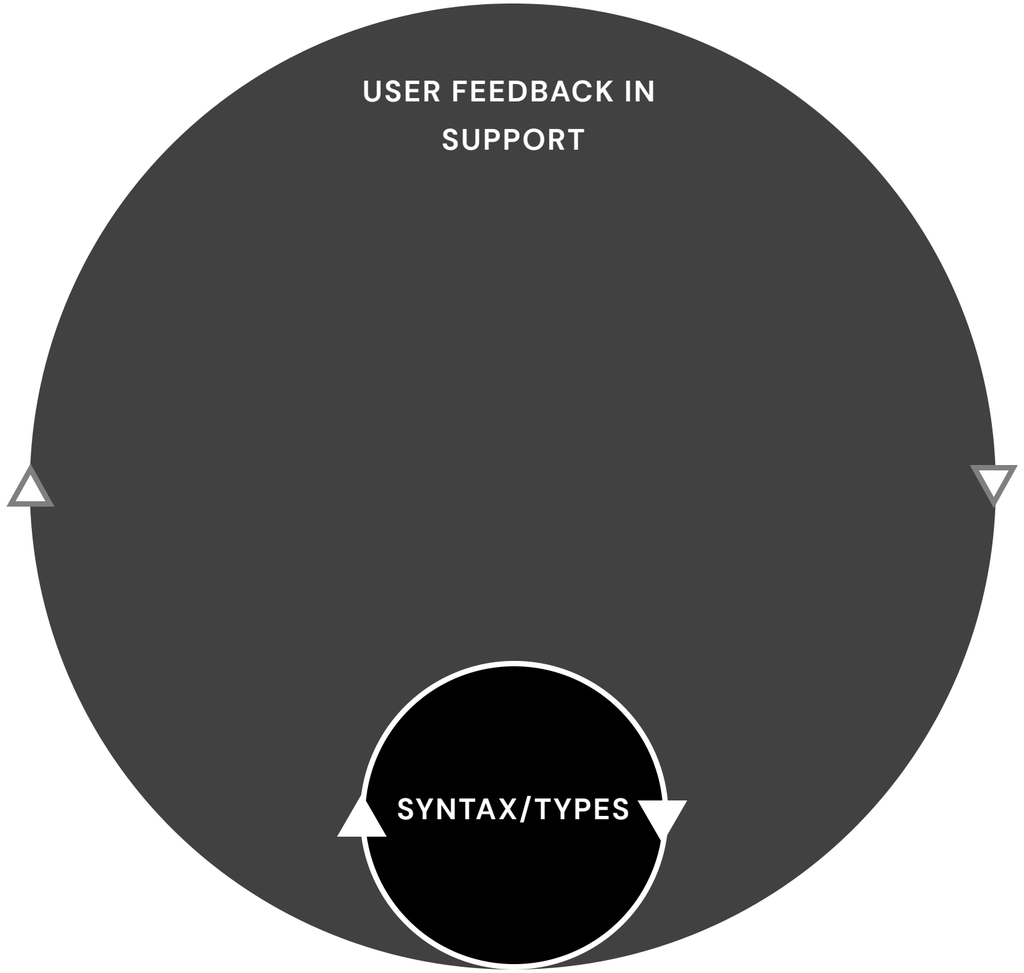
Why Unit Test Anyway?

Feedback cycles



User feedback
Has the most context, comes slow, late and already generated frustration

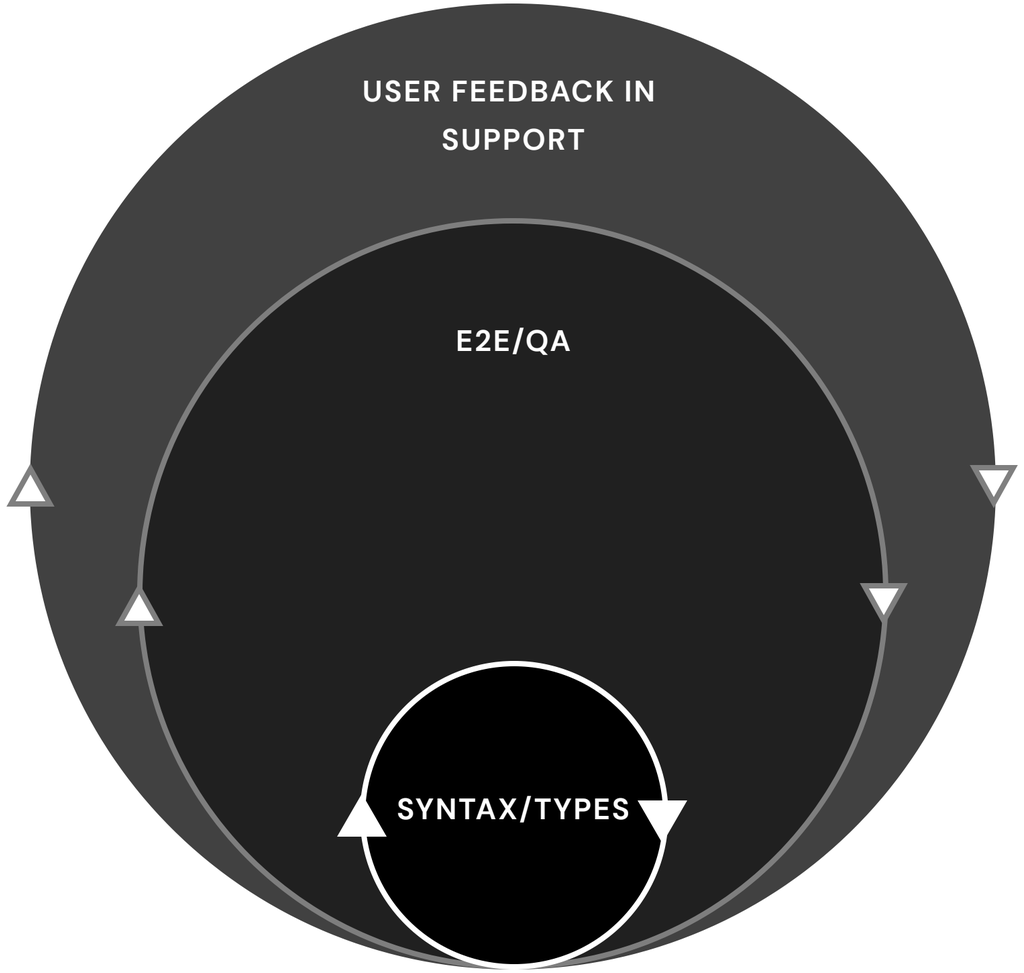
Feedback cycles



User feedback
Has the most context, comes slow, late and already generated frustration

Syntax/Types
Is instant, does not guarantee the code works as it should

Feedback cycles

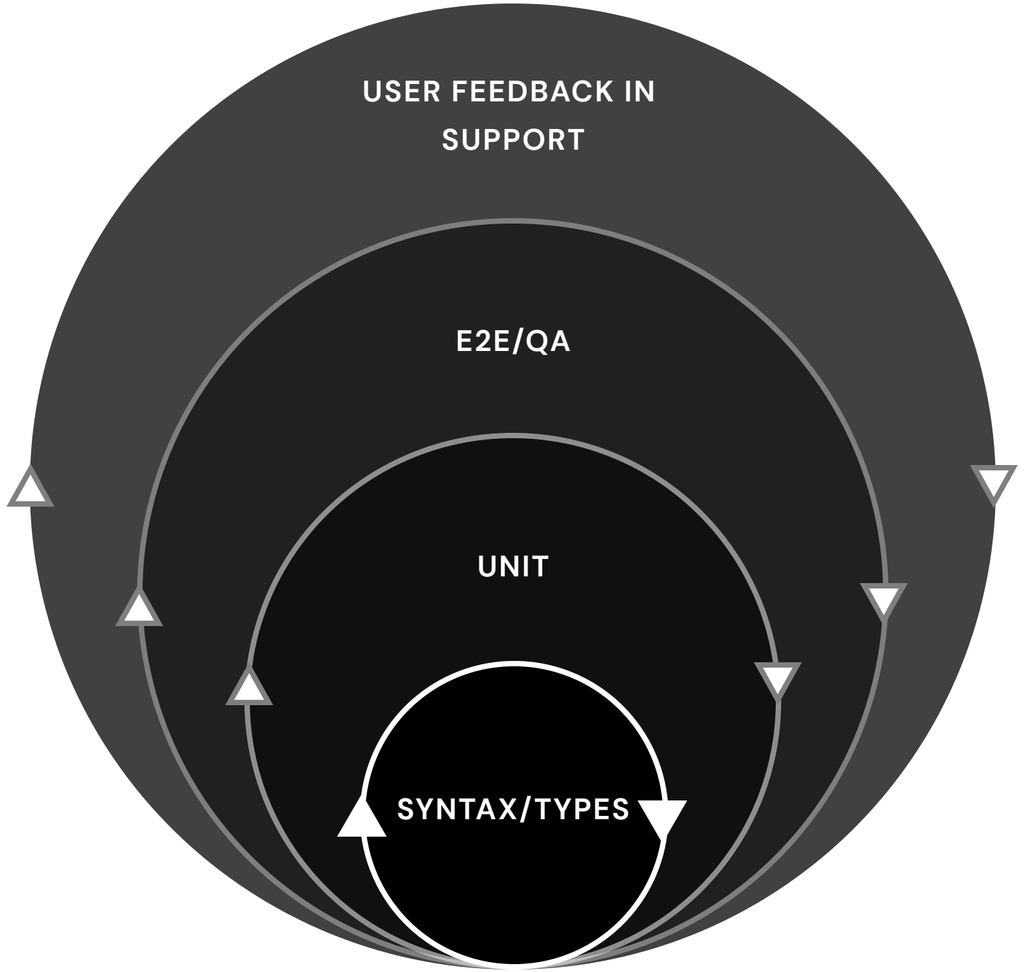


User feedback
Has the most context, comes slow, late and already generated frustration

End 2 end testing / QA
code did not reach the users, we got feedback, but it was slow as well

Syntax/Types
Is instant, does not guarantee the code works as it should

Feedback cycles

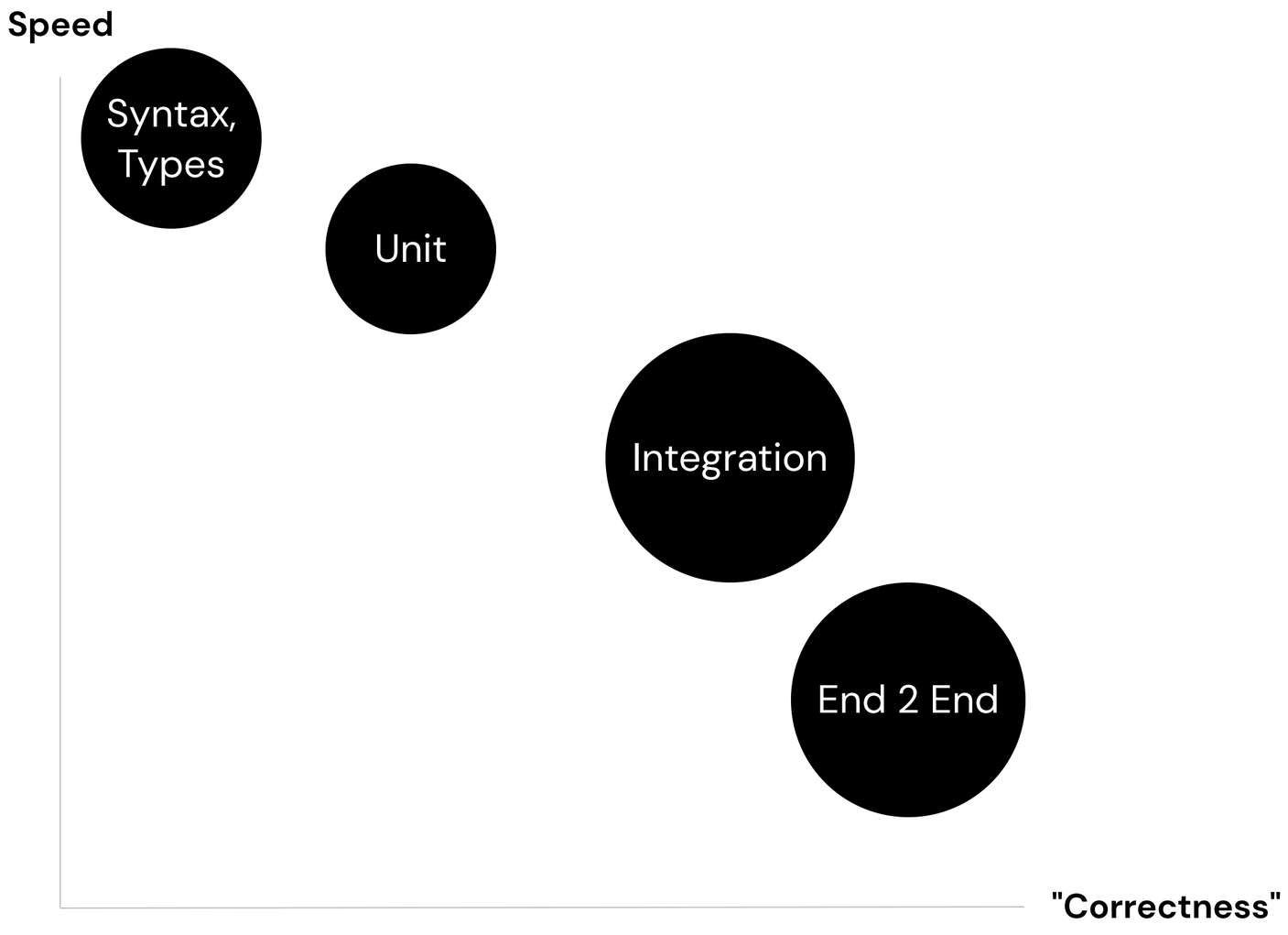


User feedback
Has the most context, comes slow, late and already generated frustration

End 2 end testing / QA
code did not reach the users, we got feedback, but it was slow as well

Unit tests
very fast feedback, but not guaranteed that the user will be satisfied

Syntax/Types
Is instant, does not guarantee the code works as it should



Goals

Speed

Isolation

Coverage

```
import TwitterClient from 'twitter'
import isPostValid from '~/path/to/isPostValid'

export default function postToTwitter(text) {
  if (!isPostValid({ text }))
    throw new Error('invalid post')

  const payload = makeTwitterPayload(text)
  TwitterClient.post(payload)

  return true
}

function makeTwitterPayload() {
  // ...snip
}
```

Approach: Not dependency inject

Pros:

- tests integration between units as well

Cons:

- complex dependency graph, which can be hard to isolate
- some things need to be isolate (eg: twitter client)

```
import postToTwitter from './postToTwitter'

describe('postToTwitter', ()=> {
  it('throws an error if post is not valid', ()=> {
    const veryLongPost = 'x'.repeat(500)

    assert.throws(()=> {
      postToTwitter(veryLongPost)
    }, /invalid post/)
  })
})
```

Approach: explicit arguments

Pros:

- very simple in implementation and understanding

Cons:

- function signature is cluttered

```
export default function postToTwitter(content, deps = {isPostValid}) {
  if (!deps.isPostValid({ content }))
    throw new Error('invalid post')

  // ...
}
```

```
import postToTwitter from './postToTwitter'

describe('postToTwitter', ()=> {
  it('throws an error if post is not valid', ()=> {
    const falseValidator = ()=> false

    assert.throws(()=> {
      postToTwitter('test', { isPostValid: falseValidator })
    }, /invalid post/)
  })
})
```

Approach: stubs

Pros:

- function does not declare its dependencies

Cons:

- default exports cannot be stubbed in sinon
- stubs house-keeping code in tests

```
import TwitterValidation from '~/path/to/TwitterValidation'

export default function postToTwitter(content) {
  if (!TwitterValidation.isPostValid({ content }))
    throw new Error('invalid post')

  // ...
}
```

```
import postToTwitter from './postToTwitter'
import TwitterValidation from '~/path/to/TwitterValidation'

describe('postToTwitter', ()=> {
  const sandbox = sinon.createSandbox();

  afterEach(()=> {
    sandbox.restore()
  })

  it('throws an error if post is not valid', ()=> {
    sandbox.stub(TwitterValidation, 'isPostValid').returns(false)

    assert.throws(()=> {
      postToTwitter('test')
    }, /invalid post/)
  })
})
```

Approach: factories

Pros:

- factories can encapsulate very complex logic

Cons:

- indirection
- implementation has test-related code nearby

```
const dummyValidator = ()=> true

export function makePostToTwitter(options = {validatorFunc: isPostValid}) {
  const isPostValid = validatorFunc ?? (process.env.NODE_ENV === 'test' ?
dummyValidator : isPostValid)

  return (text)=> {
    if (!options.validatorFunc(text))
      throw new Error('invalid post')

    // ...
  }
}

makePostToTwitter()('hello world')
```

```
import makePostToTwitter from './postToTwitter'

describe('postToTwitter', ()=> {
  it('throws an error if post is not valid', ()=> {
    const falseValidator = ()=> false

    assert.throws(()=> {
      makePostToTwitter({validatorFunc: falseValidator})('test')
    }, /invalid post/)
  })
})
```

Conclusion

andrei@planable.io

End

Isolation