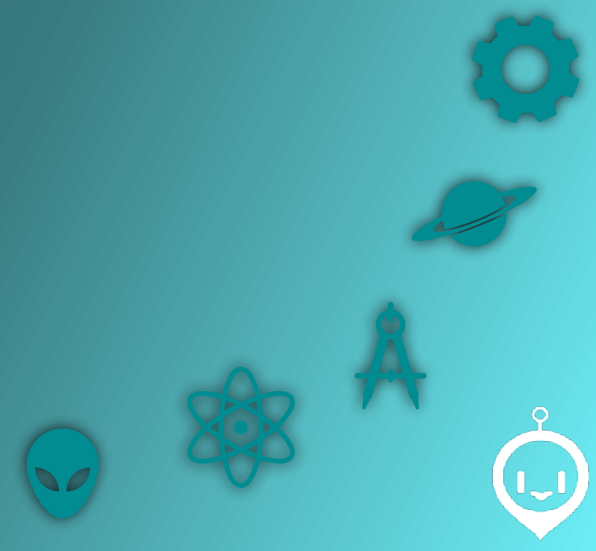
Object-oriented design patterns

Chris Gerpheide, CTO Bespoke



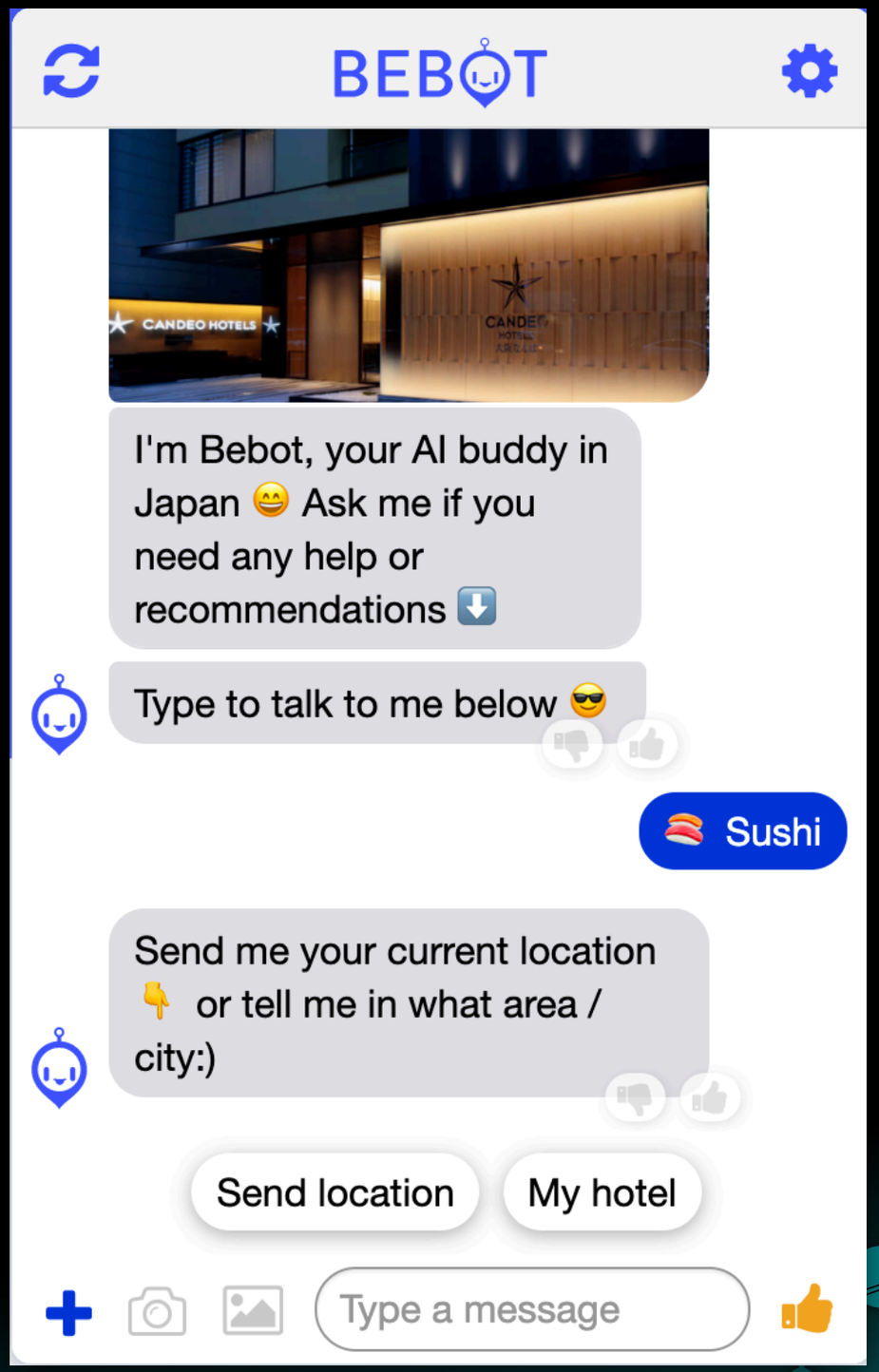
chris@be-spoke.io @phoxicle

2020-01-29

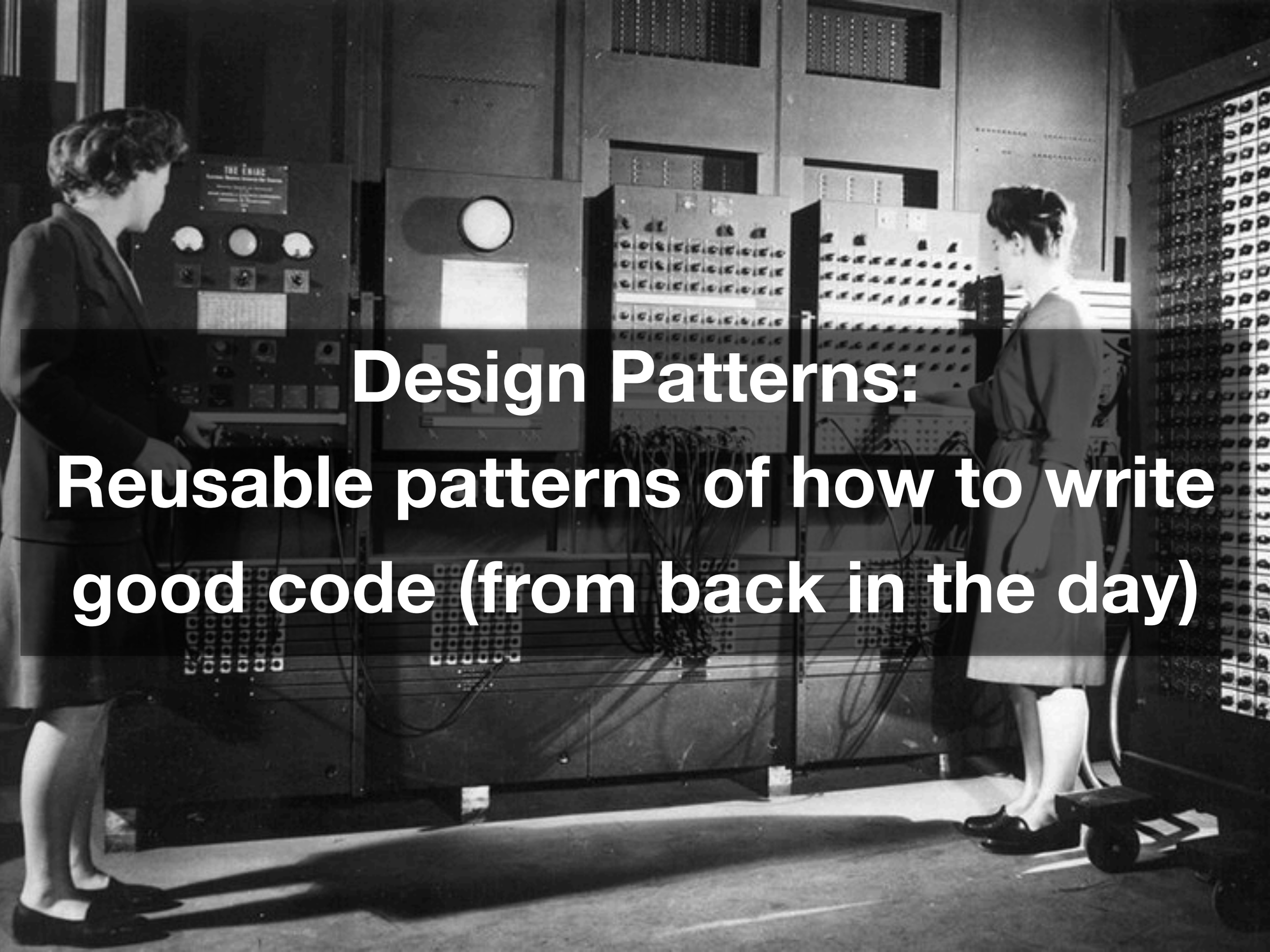
Code Chrysalis Miniconf #9

Bebot

- Chatbot for tourists
- Clients include Narita airport, Tokyo Station, JP gov't (disaster relief), Tokyo metro
- Software and Machine Learning teams



https://www.smithjournal.com.au/



Design Patterns:

Reusable patterns of how to write good code (from back in the day)

Focus on the ***UX*** of your code

- Spend time on the code you change a lot
- …And write a lot of tests

Object-oriented design patterns

- Builder
- Factory
- Facade
- Decorator

Builder (1/4)

Simplify creation of complex objects

```
query = "SELECT " + t1_col1 + "," + t1_col2 + "," +
    t2_col1 + " FROM " + t1_name  + " JOIN " +
    t2_name + " ON (" + t1_idcol + " = " + t2_idcol +
    ") ORDER BY " + t1_col1
```

```
query = "SELECT {},{},{} FROM {} JOIN {} ON ({} = {}) ORDER BY {}"
    .format(t1_col1, t1_col2, t2_col1, t1_name,
        t2_name, t1_idcol, t2_idcol,  t1_col1)
```

```
query = QueryBuilder(t1_name)
        .columns(t1_col1, t1_col2, t2_col1)
        .join(t2_name, t1_idcol, t2_idcol)
        .order_by(t1_col1)
        .build()
```

Keeps track of state

Return `self` to chain

```
class QueryBuilder:

    def __init__(self, table):
        self.table = table
        self.cols = []
        self.joins = []
        self.orderbys = []

    def columns(self, *args):
        self.cols = ', '.join(args)
        return self

    def join(self, other_table, col, other_col):
        self.joins.append('JOIN {} ON {}={}'.format(other_table, col, other_col))
        return self

    def order_by(col):
        self.orderbys.append(col)
        return self
```

Validation

Merge all the state

```
class QueryBuilder:
    ...

    def build():
        # validation
        if not self.cols:
            raise Exception('Must specify columns')
        self.cols = unique(self.cols)

        # build
        q  = 'SELECT {} '.format(','.join(self.columns))
        q += 'FROM {} '.format(self.table)
        if self.joins:
            q += '{} '.format(' '.join(self.joins))
        if self.orderbys:
            q += ' ORDER BY {} '.format(','.join(self.orderbys))
        return q
```

Builder

```
query = QueryBuilder(t1_name)
        .columns(t1_col1, t1_col2, t2_col1)
        .join(t2_name, t1_idcol, t2_idcol)
        .order_by(t1_col1)
        .where(t1_col1, value1)
        .where(t2_col2, value2)
        .join(t3_name, t1_idcol, t3_idcol)
        .build()
```

Factory (2/4)

Interface for creating related objects

```python
original_text = 'Hello! how are you todya? :)'

answer = find_answer(original_text)
if not answer:
    sanitized_text = make_ascii(original_text.lower())
    answer = find_answer(sanitized_text)
if not answer:
    answer = find_answer(typo_correct(original_text))
if not answer:
    san_and_typo_text = typo_correct(sanitized_text)
    answer = find_answer(san_and_typo_text)

return answer
```

```python
original_text = 'Hello! how are you todya? :)'

text_factory = TextFactory(original_text)

answer = find_answer(original_text)
if not answer:
    answer = find_answer(text_factory.sanitized())
if not answer:
    answer = find_answer(text_factory.typo_corrected())
if not answer:
    answer = find_answer(text_factory.sanitized_and_typo_corrected())

return answer
```

```python
class TextFactory:

    def __init__(self, text):
        self.original = text

    def sanitized(self):
        return make_ascii(self.original.lower())

    def typo_corrected(self):
        return typo_correct(self.raw)

    def sanitized_and_typo_corrected(self):
        return typo_correct(self.sanitized())
```

Factory

```
original_text = 'Hello! how are you todya? :)'

text_factory = TextFactory(original_text)

answer = find_answer(original_text)
if not answer:
    answer = find_answer(text_factory.sanitized())
if not answer:
    answer = find_answer(text_factory.typo_corrected())
if not answer:
    answer = find_answer(text_factory.sanitized_and_typo_corrected())

return answer
```

Facade (3/4)

Unified, higher-level interface to another system

```
s3.write('some_s3_bucket', 'photos/user1234/photo.jpg.gz', gzip(photo))
youtube_api.put_video('chris', 'Video for User 1234', 'video1234.mp4')
s3.write('some_s3_bucket', 'meta/user1234/metadata.json',
         json.dumps(metadata))

...

photo = gunzip(s3.read('some_s3_bucket', 'photos/user1234/photo.jpg.gz'))
video = youtube_api.get_video('chris', 'Video for User 1234')
metadata = json.loads(s3.read('some_s3_bucket', 'meta/user1234/metadata.json'))

...
```

```
user_data_facade = UserDataFacade(1234)

user_data_facade.put_photo(photo)
user_data_facade.put_video(video)
user_data_facade.put_metadata(metadata)
...
photo = user_data_facade.get_photo()
video = user_data_facade.get_video()
metadata = user_data_facade.get_metadata()
```

```python
class UserDataFacade:

    S3_BUCKET = 'some_s3_bucket'
    YOUTUBE_ACCOUNT = 'chris'

    def __init__(self, user_id):
        self.user_id = user_id

    def get_photo(self):
        return gunzip(s3.read('some_s3_bucket',
            'photos/user{}/photo.jpg.gz'.format(self.user_id)))

    def put_photo(self, corpus):
        s3.write('some_s3_bucket',
            'photos/user{}/photo.jpg.gz'.format(self.user_id), gzip(photo))

    def get_video(self):
        return youtube_api.get_video('chris',
            'Video for User {}'.format(self.user_id))

    def put_video(self):
        youtube_api.put_video('chris', 'Video for User {}'.format(self.user_id),
            'video{}.mp4'.format(self.user_id))

    def get_metadata(self):
        return json.loads(s3.read('some_s3_bucket',
            'meta/user{}/metadata.json'.format(self.user_id)))

    def put_metadata(self, classifier):
        s3.write('some_s3_bucket', 'meta/user{}/metadata.json'.format(self.user_id),
            json.dumps(metadata))
```

Different services

Handles
file names
+ types

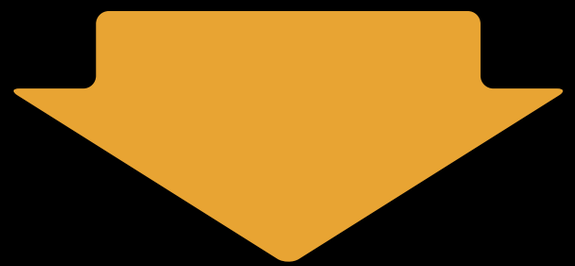
Facade

```
user_data_facade = UserDataFacade(1234)

user_data_facade.put_photo(photo)
user_data_facade.put_video(video)
user_data_facade.put_metadata(metadata)
...
photo = user_data_facade.get_photo()
video = user_data_facade.get_video()
metadata = user_data_facade.get_metadata()
```

Decorator (4/4)

Attach additional responsibilities to an object dynamically

```python
def upload():
    start_time = time()
    upload_video()
    print("Finished uploading in {} secs".format(time() - start_time ))
```



```python
@timer
def upload():
    upload_video()
```

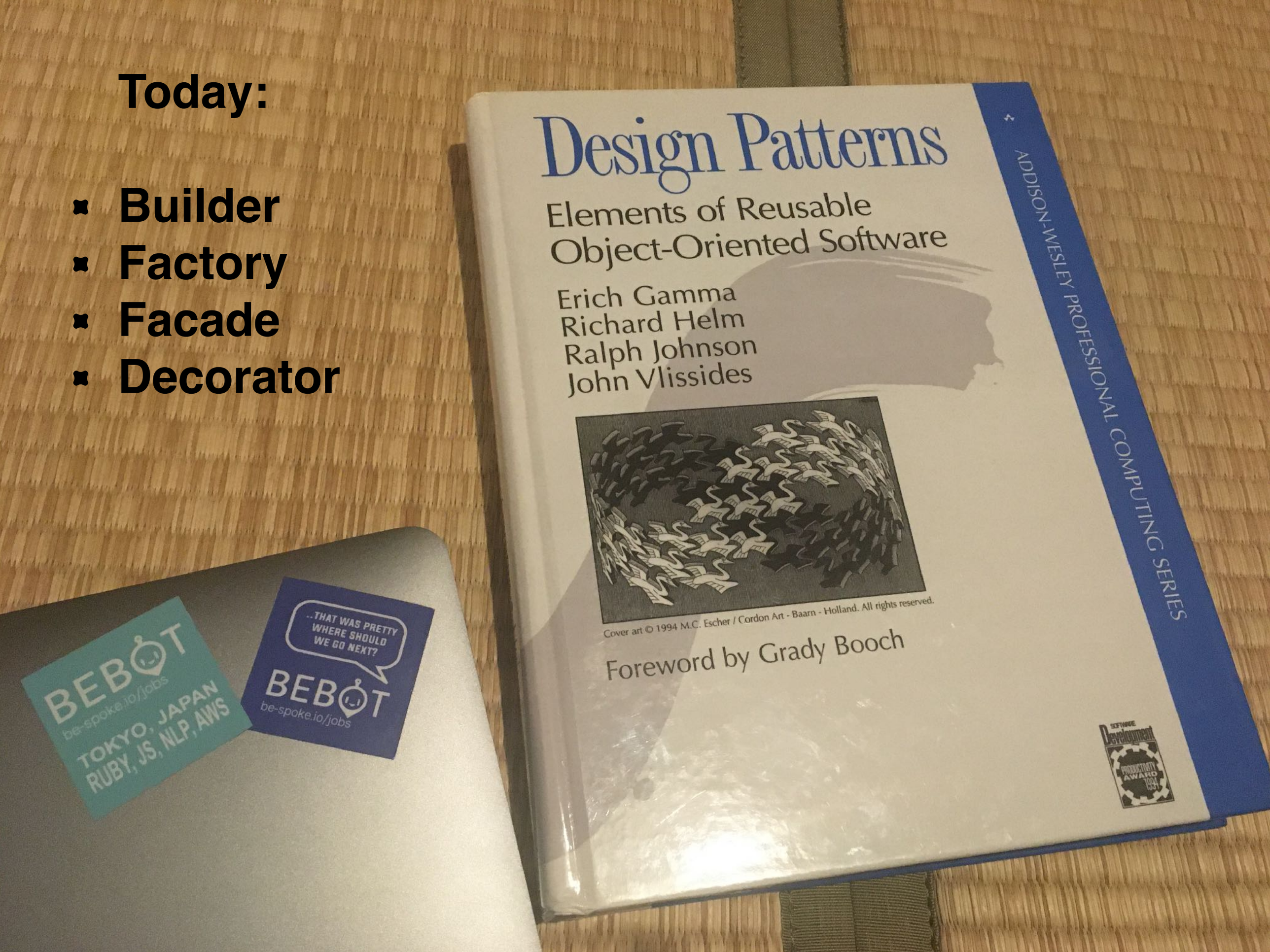
```python
def timer(func):

    @functools.wraps(func)
    def wrapper_timer(*args, **kwargs):
        start_time = time()
        value = func(*args, **kwargs)
        print("Finished {} in {} secs"
              .format(func.__name__, time() - start_time))
        return value

    return wrapper_timer
```

Decorator

```
@timer
def upload():
    upload_video()
```

Today:

- **Builder**
- **Factory**
- **Facade**
- **Decorator**



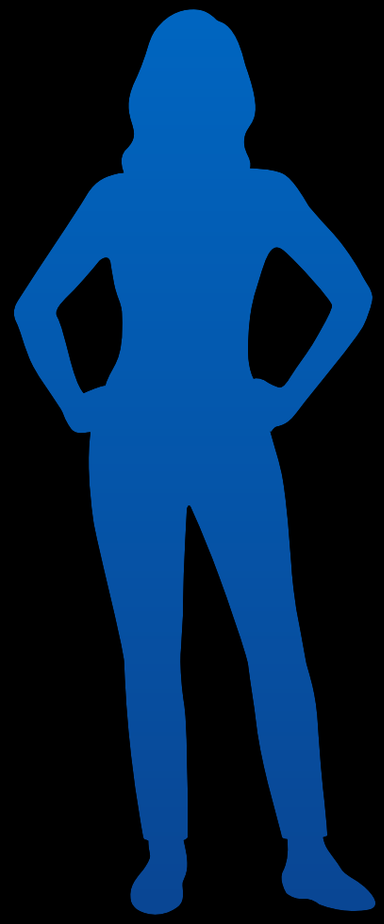
Focus on the UX of your code

```python
@timer
def upload(user_id):
    ...
    user_video = user_data_facade.get_video()

    user_video_factory = VideoTreatmentFactory(user_video)

    composed_video = CompositeVideoBuilder()
        .with_video(video_factory.black_and_white())
        .with_video(code_chrysalis_promo_video)
        .include_cut_scenes()
        .trim_to(seconds=45)
        .build()
```

That code you wrote was very clever…

Thanks!

That wasn't a compliment



Senior programmer

Junior programmer

BESPOKE

Focus on the UX of your code

```python
@timer
def upload(user_id):
    ...
    user_video = user_data_facade.get_video()

    user_video_factory = VideoTreatmentFactory(user_video)

    composed_video = CompositeVideoBuilder()
        .with_video(video_factory.black_and_white())
        .with_video(code_chrysalis_promo_video)
        .include_cut_scenes()
        .trim_to(seconds=45)
        .build()
```

Object-oriented Design Patterns

Chris Gerpheide, CTO Bespoke

chris@be-spoke.io @phoxicle

Optional: Add caching

```
class TextFactory:

    def __init__(self, text):
        self.original = text

    def sanitized(self):
        if not self._sanitized:
            self._sanitized = make_ascii(self.original.lower())

    def typo_corrected(self):
        if not self._typo_corrected:
            self._typo_corrected = typo_correct(self.original)
        return self._typo_corrected

    def sanitized_and_typo_corrected(self):
        if not self._sanitized_and_typo_corrected:
            self._sanitized_and_typo_corrected = typo_correct(self.sanitized())
        return self._sanitized_and_typo_corrected
```