API Design Lessons Learned: Enterprise to Startup

Mohamed El-Geish
g@workfit.io

eva | workfit

WARNING

- This presentation may contain content that contradicts how your company designs APIs
- Startups are nimble and daring; don't try this at ~~home~~ work (or do try it)!
- YMMV


eva
workfit
WHAT ARE WE BUILDING?
ENTERPRISE VOICE AI

eva CAPABILITIES

Automatic
Dial-in

Takes Interactive
Commands

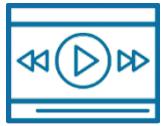
Transcribes
Key Highlights

Indexed and
Searchable
Meetings

Interactive
Meeting
Dashboard

Interactive
Word Cloud

Share Meeting
Highlights

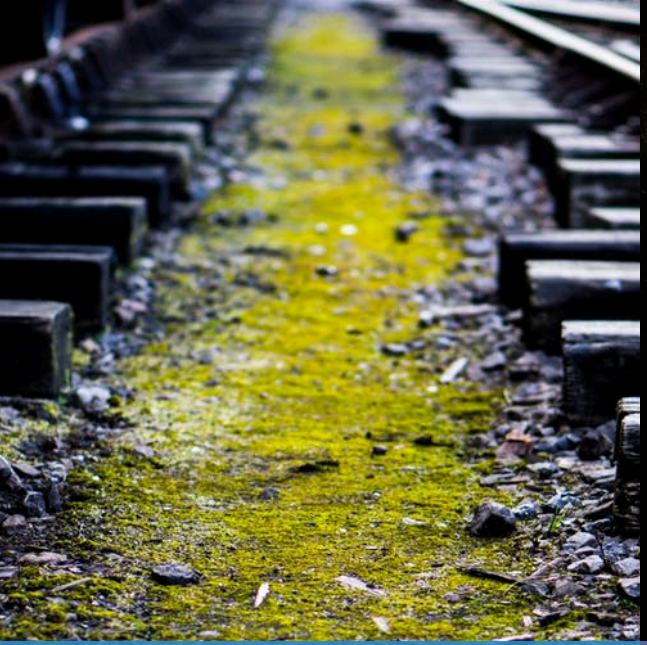


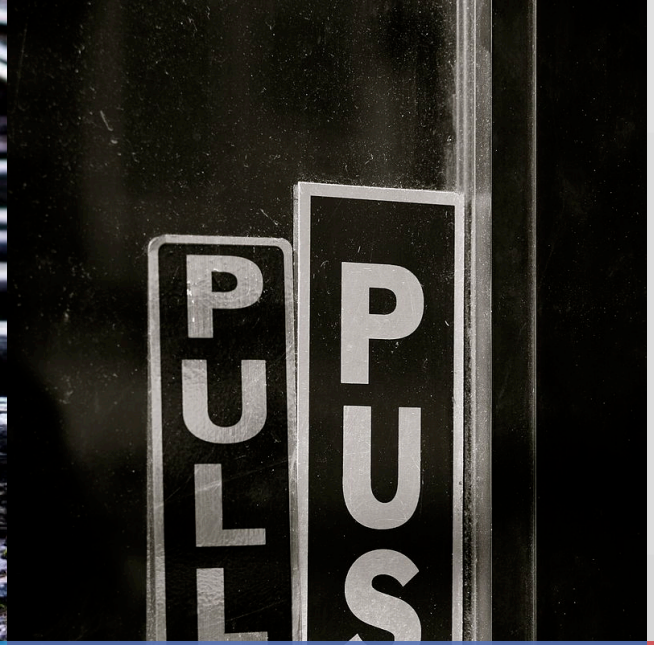
Highlight
Reel

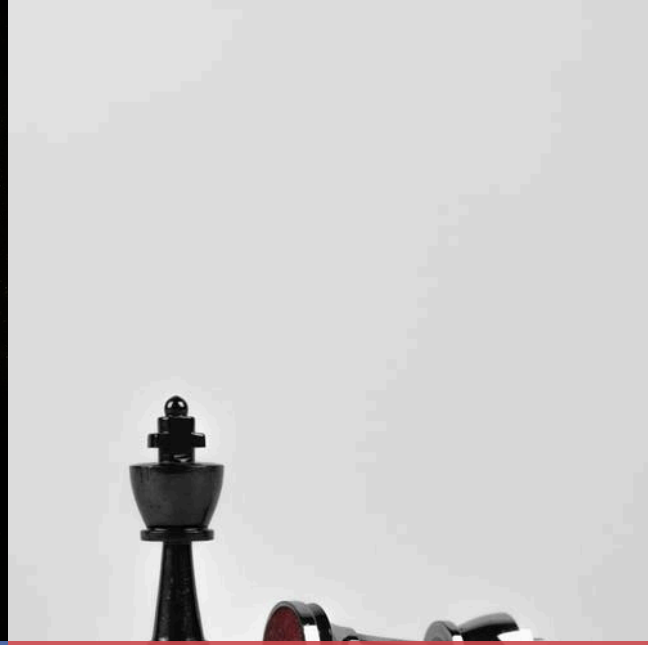
Meeting
Recap Email

LET’S TALK API DESIGN

WHY API DESIGN MATTERS?







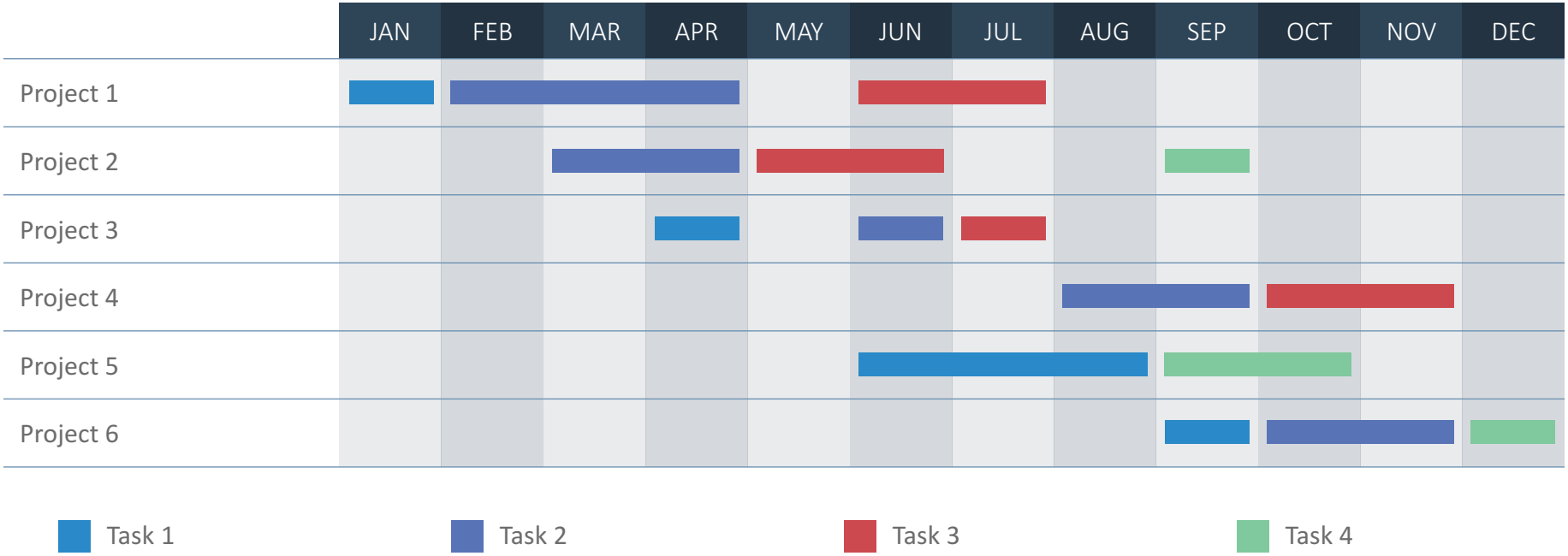
BECAUSE

BAD API DESIGN

IS COSTLY!

workfit

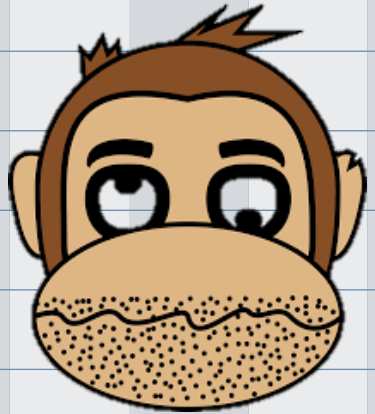
TIMELINE WHEN API DESIGN IS GOOD



workfit

TIMELINE WHEN API DESIGN IS BAD

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
Project 1												
Project 2												
Project 3												
Project 4												
Project 5												
Project 6												



Task 1 Task 2 Task 3 Task 4

workfit

EXAMPLE: A TALE OF TWO QUEUES

```
// We used to use a FIFO SQS queue to send messages:
request, _ := q.SendMessageRequest(&sqs.SendMessageInput{
    QueueUrl: aws.String(queueURL),
    MessageBody: aws.String(messageBody),
    MessageGroupId: aws.String(groupID),
    MessageDeduplicationId: aws.String(dedupeID),
})
return request.Send()

// Then we switched that code to use a standard (non-FIFO) queue.
// Expectation: The API clearly denotes a FIFO-specific interface.
```

EXAMPLE: A TALE OF TWO QUEUES

```
// Reality: latent errors due to ambiguous API overloading;
// we can’t use the last two arguments for standard queues!
// This parameter applies only to FIFO (first-in-first-out) queues.
// ... [many lines later] ...
// MessageGroupId is required for FIFO queues.
// You can't use it for Standard queues.

// One way to mitigate this is to create a FIFO-specific API:
request, _ := q.SendFIFOMessageRequest(&sqs.SendFIFOMessageInput{
    MessageGroupId: aws.String(groupID),
    MessageDeduplicationId: aws.String(dedupeID),
...
```

LESSON 0: IMITATE THE GREAT (LEARN)

PERCEPTUAL LEARNING

- **Study:** Non-pilots did better than experienced pilots; PL works for other complex fields.*
- **Cognitive Science:** PL accelerates gaining expertise via pattern recognition.**
- **SMDMTM:** See ~~one~~ many; do ~~one~~ many; teach ~~one~~ many — we require many high-quality examples for high SNR.***

BIG COMPANIES = GREAT FOR LEARNING

1. Ample time and resources
2. Scrutinized processes and reviews
3. Access to a ton of code and designs
4. Formal coaching and training
5. Abundance of talent and expertise

THE WORKFIT WAY

LEARNING AS A TENET

- Humans are the most important factor in the success of software.
- We learn to make new mistakes.
- We invest in learning and coaching: long-term ROI.
- Accelerated learning => high iteration velocity.

LEARNING TO DO
DOING TO LEARN

LET’S TALK CODE

EXAMPLE: PACKAGE MUST

```
// We noticed a pattern when declaring main package variables
// We also noticed a pattern in Go's standard library:
// MustCompile is like Compile but panics if the expression cannot be
// parsed. It simplifies safe initialization of global variables holding
// compiled regular expressions.
func MustCompile(str string) *Regexp {
  regexp, error := Compile(str)
  if error != nil {
      panic(`regexp: Compile(` + quote(str) + `): ` + error.Error())
  }
  return regexp
}
```

EXAMPLE: PACKAGE MUST

```
// So we imitated the same pattern into package must:
// CreateMeetingsClient wraps fabric.NewMeetingsClient
// with default parameters.
func CreateMeetingsClient() fabric.MeetingsClient {
    client, err := fabric.NewMeetingsClient(
        context.TODO(), fabric.DefaultClientConfig)
    if err != nil {
        reportPanic(err)
    }
    return client
}
var meetingsClient = must.CreateMeetingsClient() // how it's called
```

ANOTHER EXAMPLE: PACKAGE ERRORS

```
// Visual Studio TFS errors have IDs to reference them in docs*;
// free-form errors are hard to map into a single TSG/failure mode;
// our errors package allows for consistency, reuse, strong contracts,
// brevity, testability, error handling hooks, and localizability.

const wf11200 = `WF11200: HTTP response status code was not 2xx`

// WF11200 occurs when an HTTP response has a status code other than 2xx.
func WF11200(response interface{}) error {
    log.Error(wf11200, "response", response)
    return newError(wf11200)
}
```

LESSON 1: FIND YOUR NORTH STAR

“As to the methods there may be a million and then some, but principles are few. The man who grasps principles can successfully select his own methods. The man who tries methods, ignoring principles, is sure to have trouble.”

RALPH WALDO EMERSON

WHAT? WHY? HOW?

- A north star guides your decision-making processes including API design choices.
- In ever-changing environments, one can get lost and distracted by myopic goals.
- Align design goals with business goals; we don’t design in a vacuum.
- Find your treasure and guard it well.

KEY CHALLENGES STARTUPS & NEW PROJECTS FACE

- **A blank canvas:** an overwhelming number of decisions to make -> decision fatigue.
- **Velocity is life; grow fast or die slow:** "If a software company grows at [20% annually], it has a 92 percent chance of ceasing to exist within a few years."*
- **Security:** it's inconvenient; it's everyone's responsibility; and it can slow us down.

THE WORKFIT WAY

SETTING GOALS & GUIDING PRINCIPLES

- Scope: short- vs. long- term.
- Agreeing on guiding principles -> a decision-making framework -> conflict-resolution mechanism.
- We strive to foster wisdom and autonomy.

GOALS AND GUIDING PRINCIPLES

- **Short-term:** security, correctness, iteration velocity, availability, performance, throughput, scalability, and maintainability.
- **Long-term:** security, correctness, availability, throughput, performance, scalability, maintainability, and iteration velocity.
- **Balance:** Design with scalability and maintainability in mind; trade off only when necessary.
- **Methods:** encryption, ACLs, cyber hygiene, CI, testing, SOA, A/B testing, tracking and telemetry, KISS, etc.

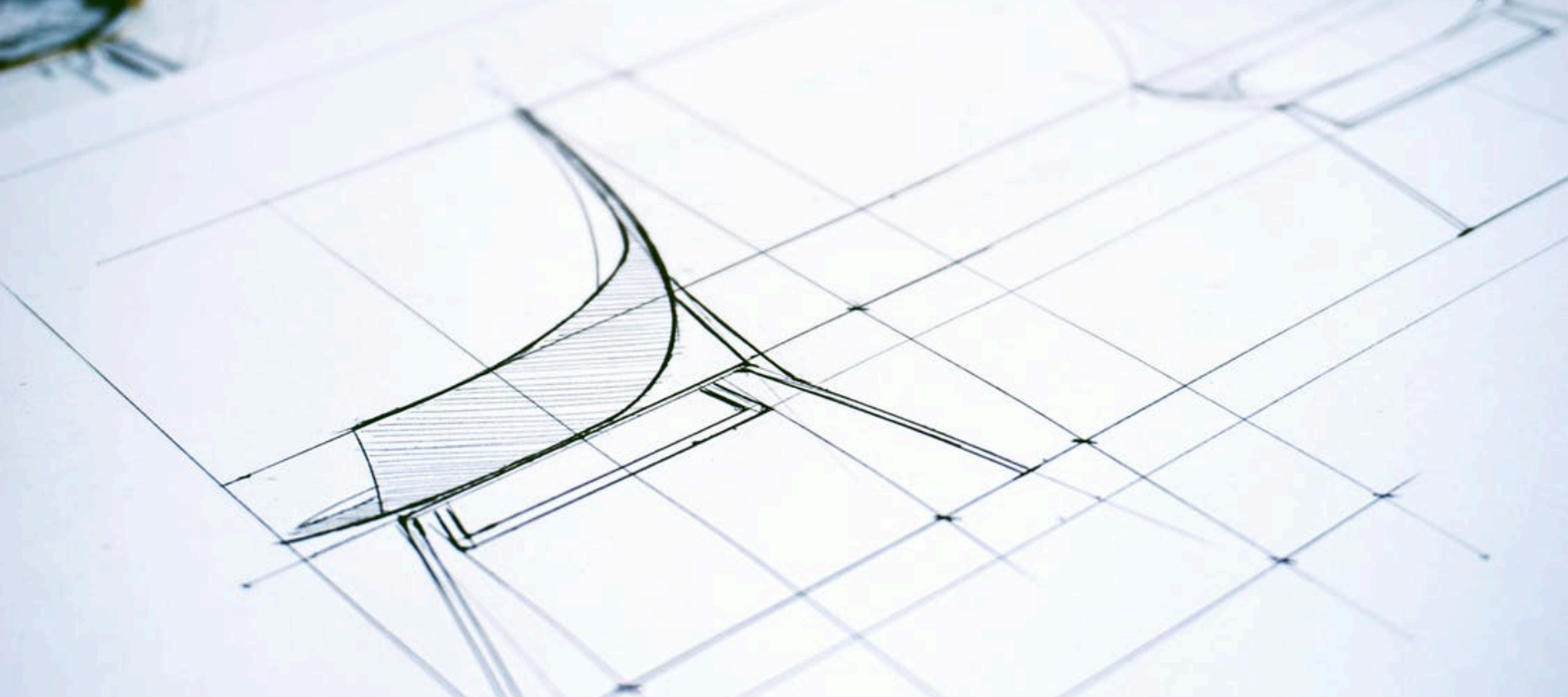
SHORT-TERM GUIDING PRINCIPLES IN ACTION

- **Data Formats:** Binary or textual?
- **Contracts:** Whether or not to enforce schemas?
- **Programming Languages:** Why Go, C++, Python, and Java?
- **Investing in CI:** Travis, Docker, and DC/OS.
- **Rich Logging:** verbose and structured; multi-engine; and secure.

EXAMPLE: LIFECYCLE OF A MICROSERVICE

LET'S TALK MORE ABOUT CHALLENGES

- **Latent conflict of priorities:** Go is great but it had its issues (e.g. ICS parser).
- **Building for scale:** balancing TTM with future projections of scalability.
- **Susceptibility to tribal knowledge syndrome:** how to share knowledge & move fast? e.g., `log.Error(wf11200, "response", response) // what's "response"?`
- **Many choices:** green-field projects with high degrees of freedom; e.g., we moved from Kubernetes to DC/OS because of:
  - Better support of security and stateful solutions
  - Stability (on AWS)
  - GPU Time-sharing



LESSON 2: DESIGN FOR HUMANS

LEARN ABOUT ALL SORTS OF DESIGN

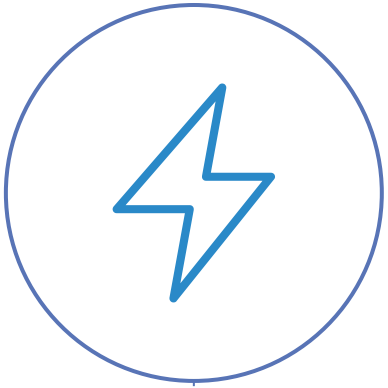
- Good API design qualities transcend code.
- Industrial design cares about interfacing with a physical product.
- User-centered design focuses on usability.
- Many analogies to draw among all sorts of design.

TEN PRINCIPLES OF GOOD DESIGN BY DIETER RAMS

1. Good Design Is **Innovative**
2. Good Design Is **Useful**
3. Good Design Is **Aesthetic**
4. Good Design Is **Understandable**
5. Good Design Is **Unobtrusive**
6. Good Design Is **Honest**
7. Good Design Is **Long-lasting**
8. Good Design Is **Thorough**
9. Good Design Is **Eco-friendly**
10. Good Design Is **Minimal**

PUT THE HUMAN FIRST!

Good API design is about
communicating well to humans



If the human gets it,
things will work quicker & better

“Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.”

DONALD KNUTH

EXAMPLE: PACKAGE ASSERT

```
// Java projects at LinkedIn are tested using TestNG, JUnit, & AssertJ:
org.testng.Assert.assertEquals(x, y) // actual, expected
org.junit.Assert.assertEquals(x, y) // expected, actual -> cognitive load
Assertions.assertThat(x).isEqualTo(y); // better

// Package assert* allows for easy & consist UT+DDT and test hook checks
if !assert.For(t).ThatActual(x).Equals(y).Passed() {
    analyze(x, y)
}
assert.For(t).ThatActual(x).Equals(expected).ThenRunOnFail(analyze)
assert.For(t).ThatActual(x).Equals(expected).ThenDiffOnFail()
assert.For(t).ThatCalling(someFunc).PanicsReporting("error")
```

EXAMPLE: PACKAGE ASSERT

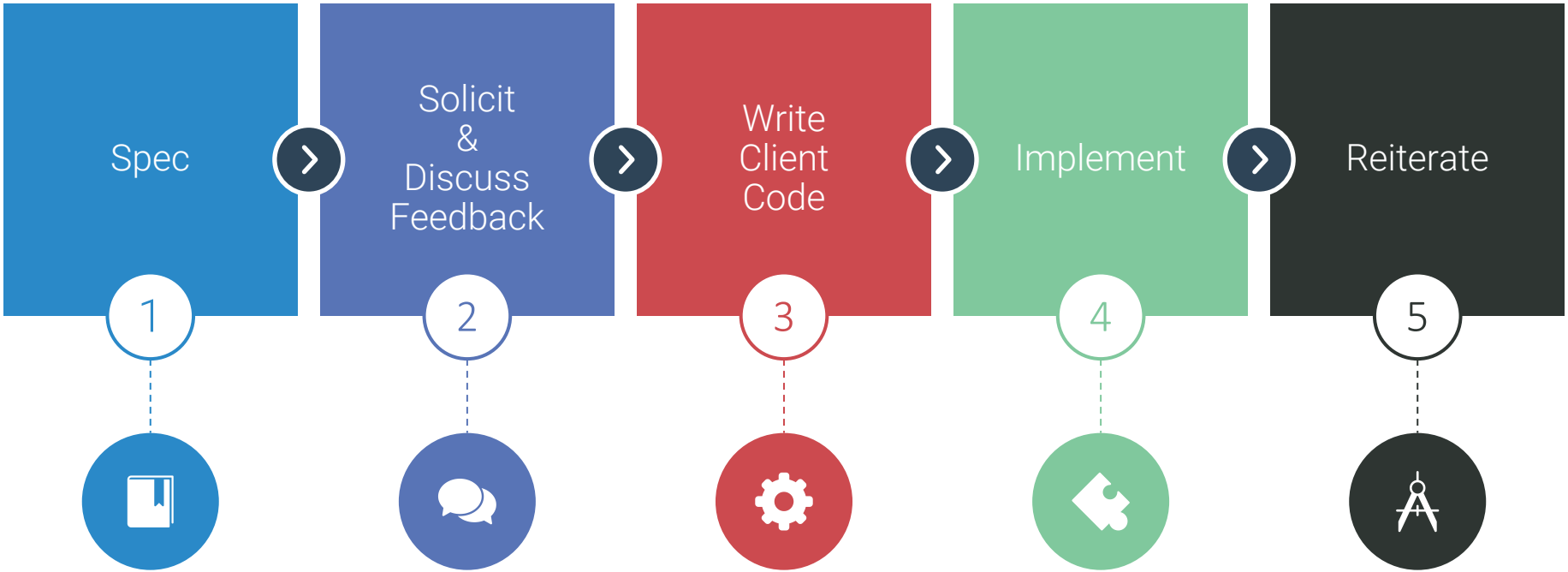
```
// Plays well with table-driven tests:
cases := []struct {
    id   string
    actual interface{}
    expected interface{}
}{
    {"different values", 42, 13},
    // ...
}
for _, c := range cases {
    assert.For(t, c.id).ThatActual(c.actual).Equals(c.expected)
} // prints test case ID in failure messages
```

EXAMPLE: PACKAGE ASSERT

```
// At Microsoft, we used the internal access modifier and friend
// assemblies for testability; at LinkedIn we used package-private
// methods and documented that they are @VisibleForTesting; at Workfit,
// using tags, instead of comments, enables us to check test hooks:
type sleeper struct {
    sleep func(time.Duration) `test-hook:"verify-unexported"`
}

// What gets exposed for testability shall not be exported!
func TestHooksAreHidden(t *testing.T) {
    assert.For(t).ThatType(reflect.TypeOf(sleeper{})).HidesTestHooks()
}
```

ANOTHER EXAMPLE: API DESIGN PROCESS



workfit


eva
workfit
FINALLY

TAKEAWAYS: 3x3

LEARNING	NORTH STAR	PEOPLE FIRST
• See	• Goals	• Accelerate
• Do	• Tenets	• Communicate
• Teach	• Methods	• Empathize


eva
workfit
WE'RE HIRING
ENTERPRISE VOICE AI

CONTACT INFO

@elgeish

g@workfit.io

www.elgeish.com

linkedin.com/in/elgeish