Choice is the Enemy of Good

William Bartlett
@ william.bartlett@zenika.com @bartlettstarman
punkstarman punkstarman

Zenika

Γειά σας!
Με λένε Γουίλιαμ.
Είμαι Βρετανός και Αμερικανός.
Μένω στη Νάντη στη Γαλλία.
Είμαι σύμβουλος στο Ζένικα.

JS

We have yet to write a single line of business logic

We have yet to write a single line of business logic

Other programming languages have similar problems.

How can we get stuff done and feel good about it?

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

— Agile Manifesto

Sustainable development is development that meets the needs of the present without compromising the ability of future generations to meet their own needs.

— Brundtland Report

Section 1

How do we choose?

Reason

That's all Folks!

Emotion

Emotion

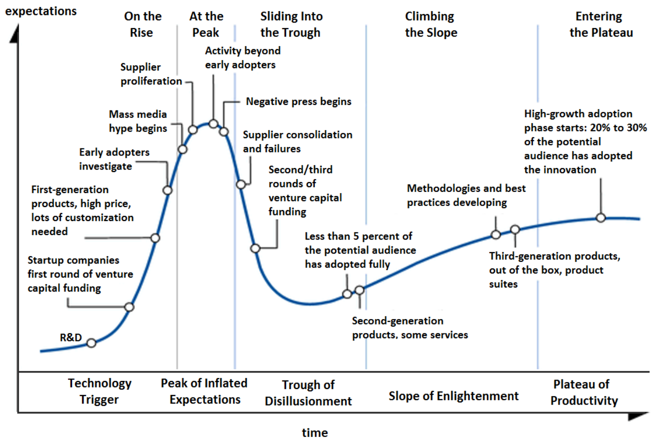
> *4. Rationalizing an emotional decision. Go was fun. It felt right. We thought: "if we're willing to adopt Go on a gut feeling, maybe other systems geeks will be too? And wouldn't it be cool if we could get them to feel that way about Docker, too?"*
>
> — Solomon Hykes

Emotion

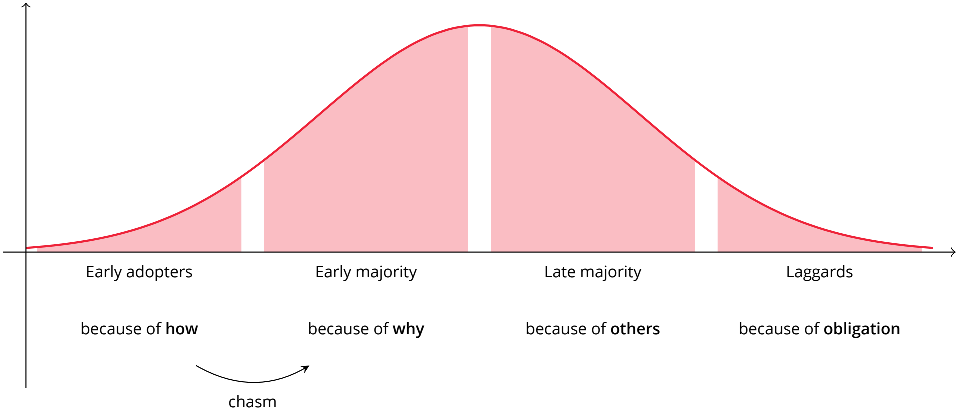
> *Reason is, and ought only to be the slave of the passions, and can never pretend to any other office than to serve and obey them.*

— David Hume. A Treatise of Human Nature

Observation


expectations
On the Rise
At the Peak
Sliding Into the Trough
Climbing the Slope
Entering the Plateau
Activity beyond early adopters
Supplier proliferation
Negative press begins
Mass media hype begins
Supplier consolidation and failures
Early adopters investigate
Second/third rounds of venture capital funding
First-generation products, high price, lots of customization needed
High-growth adoption phase starts: 20% to 30% of the potential audience has adopted the innovation
Methodologies and best practices developing
Less than 5 percent of the potential audience has adopted fully
Startup companies first round of venture capital funding
Third-generation products, out of the box, product suites
Second-generation products, some services
R&D
Technology Trigger
Peak of Inflated Expectations
Trough of Disillusionment
Slope of Enlightenment
Plateau of Productivity
time

Lifecycle



Diffusion of Innovation. Everett Rogers. 1962 — Crossing the Chasm. Geoffrey A. Moore. 1991

Luck

Luck

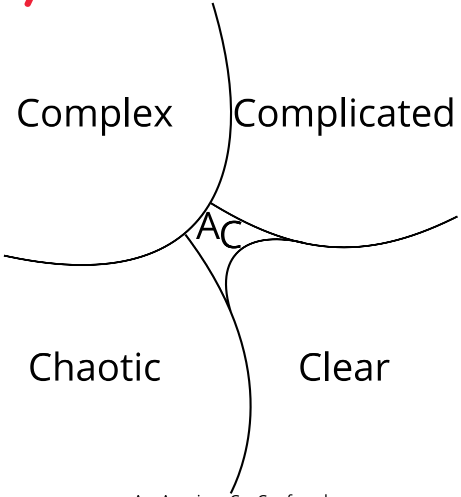
[Luck] plays a very large role in every story of success.

— Daniel Kahneman. Thinking, Fast and Slow

Section 2

Contextual Decision-Making

Cynefin



A = Aporia C = Confused

Cynefin (Κυνέβιν)

Sense-making framework created by Dave Snowden

Cynefin

Clear

Predictable cause and effect known to all

- Apply best practices
- Take time for tech watch

Examples:

- indentation, code style
- sorting algorithm

> *I have yet to see any problem, however complicated, which, when you looked at it in the right way, did not become still more complicated.*

— Anderson’s Law

Cynefin

Complicated

Predictable cause and effect known to experts

- Ask experts, rely on experience
- Use analysis expertise is lacking
- Share expertise, when needed

Examples:

- testing strategies
- security

Cynefin

Complex

Any action has unintended consequences

Examples:

- large-scale distributed systems
- organizations, teams

Cynefin

Complex

Any action has unintended consequences

- Diversity

Cynefin

Complex

Any action has unintended consequences

- Diversity
- Do the next right thing (coherence)

Cynefin

Complex

Any action has unintended consequences

- Diversity
- Do the next right thing (coherence)
- Keep options open

Cynefin

Complex

Any action has unintended consequences

- Diversity
- Do the next right thing (coherence)
- Keep options open
- Let the market decide

Cynefin

Complex

Any action has unintended consequences

- Diversity
- Do the next right thing (coherence)
- Keep options open
- Let the market decide
- Document decisions (Architecture Decision Record)

Parallel safe to fail probes

> *When trying something new, do it on a scale where failure is survivable.*

— Peter Palchinsky’s 2nd principle

Parallel safe to fail probes

> *When trying something new, do it on a scale where failure is survivable.*

— Peter Palchinsky’s 2nd principle

> *Nothing is more dangerous than an idea when it is the only one you have.*

— Emile Chartier

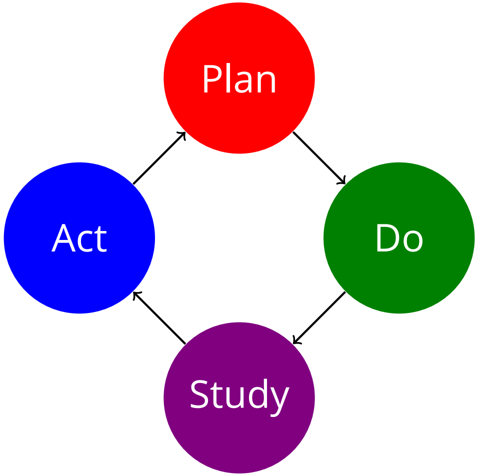
Retrospective Prime Directive

Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.
— Norm Kerth. Project Retrospectives: A Handbook for Team Review

Section 3

Plan-Do-Study-Act

Plan-Do-Study-Act



Plan-Do-Study-Act

- renders certain arguments unnecessary

Plan-Do-Study-Act

- renders certain arguments unnecessary
- pushes us to debate facts instead of fiction

Plan-Do-Study-Act

- renders certain arguments unnecessary
- pushes us to debate facts instead of fiction
- allows codebases to mature at the same rate as the team

Malleable Code

Malleability enhances PDSA

Malleable Code

Malleability enhances PDSA

- Disposability

Malleable Code

Malleability enhances PDSA

- Disposability
- Decoupling

Malleable Code

Malleability enhances PDSA

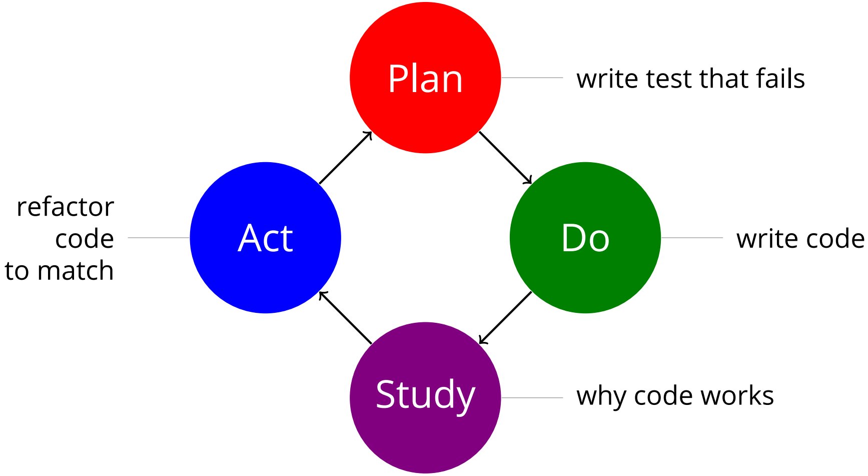
- Disposability
- Decoupling
- Test-Driven Development

What TDD is not

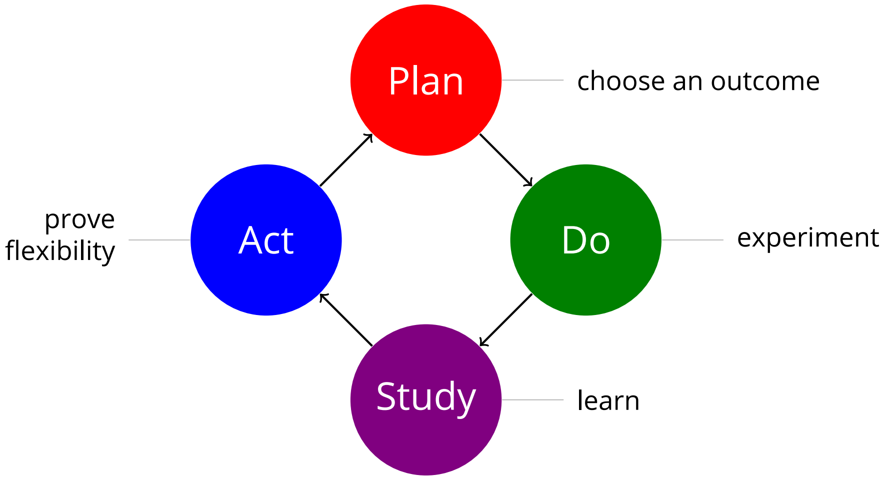
TDD is not

- writing the tests before the code
- unit testing every single function
- a substitute for other forms of testing

What TDD is


Plan
write test that fails
Do
write code
Study
why code works
Act
refactor
code
to match

What TDD accomplishes



Evolution of a Java Programmer

```java
public class Factorial {
    public long factorial(long n) {
        ...
    }
}
```

Evolution of a Java Programmer

Java 1.1 - simple and effective

```java
public long factorial(long n) {
    long result = 1;
    for (long i=1; i<=n; i++) {
        result = result * i;
    }
    return result;
}
```

Evolution of a Java Programmer

Java 1.2 - Collections

```java
public long factorial(long n) {
    List factors = new ArrayList();
    for (long i=1; i<=n; i++) {
        factors.add(i);
    }
    long result = 1;
    Iterator it = factors.iterator();
    while (it.hasNext()) {
        long factor = ((Long)it.next()).longValue();
        result *= factor;
    }
    return result;
}
```

Evolution of a Java Programmer

Java 5 - Generics and foreach loops

```java
public long factorial(long n) {
    List<Long> factors = new ArrayList<Long>();
    for (long i=1; i<=n; i++) {
        factors.add(i);
    }
    long result = 1;
    for (long factor: factors) {
        result *= factor;
    }
    return result;
}
```

Evolution of a Java Programmer

Java 7 - Parallelism

```
public long factorial(long n) {
    ForkJoinPool pool = new ForkJoinPool();
    return pool.invoke(new FactorialTask(1, n));
}
```

Evolution of a Java Programmer

Java 7 - Parallelism

```java
class FactorialTask extends RecursiveTask<Long> {
        @Override
        protected Long compute() {
            if (lo > hi) return 1L;
            if (lo == hi) return lo;

            long mid = (lo + hi) / 2L;
            FactorialTask f1 = new FactorialTask(lo, mid);
            f1.fork();
            FactorialTask f2 = new FactorialTask(mid + 1L, hi);
            return f2.compute() * f1.join();
        }
}
```

Evolution of a Java Programmer

Java 8 - Functional streams

```
public long factorial(long n) {
    return LongStream
    .rangeClosed(1, n)
    .reduce(1, (a, b) -> a * b);
}
```

Section 4

Conclusion

Choice Is the Enemy of Good

Don’t complicate things needlessly

Develop and use expert knowledge and experience

Plan-Do-Study-Act (TDD)

Do the next right thing

Keep options open