Feature Parity in 25% Dev Hours
Quinn Gil
(last talk, last day)

@TheQuinnGil

QuinnGil.com
The Story

- Rewrite of mobile apps (iOS, Android, Windows)
- Started iOS & Android
- 9m into dev (only 2/3 complete) started Windows
- Given ½ the team.
- Given only 7 months to develop the same features
- Asking the impossible - But OK - With Conditions
  - BENEVOLENT DICTATOR
  - Strict XP
  - Strict set of Technical Practices
- I’ll learn, even if I don’t succeed
  - Heard XP is great, never saw strict application
  - Heard the code practices are great, never saw in action
- Everyone agreed, and I started my dictatorship
- Result - 7m version is available in the store.
- XP & Technical Practices WORK!
- 25% of Dev hours for Feature Parity.
Why did it work?

- PO / Customer in arms length
- **Pair+ Programming**
  - Hours of discussion to remove a single line
  - Which enabled refactoring 30% of the code
  - Included PO/Customer
  - Included Design
- **Sustainable Pace / NO Crunch Time**
- **#NoEstimates**
- **Refactoring**
- **Emergent Architecture / Simple Design**
- **Testing**
- **Continuous Delivery**
- **Coding Standards**
- **Collective Ownership**
  - Devs challenge design
  - Design challenge devs

eXtreme Programming
Why did it work?

Good code invariably has small methods and small objects. Only by factoring the system into many small pieces of state and function can you hope to satisfy the “once and only once” rule. ... [N]o one thing I do to systems provides as much help as breaking it into more pieces.

Kent Beck - Smalltalk Best Practice Patterns `97
Technical Practices

- No Getters / No Setters
- `if` Only As Guard Clause
- Isolate Their Code
- Never `null`
- No `new` inline
- Composition, Not Inheritance
- <Strong opinions, loosely held>
- Be Immutable
- No Primitives
- Extract Cohesion
- No Public Statics
- Never Reflection
Good code invariably has small methods and small objects. Only by factoring the system into many small pieces of state and function can you hope to satisfy the “once and only once” rule. … [N]o one thing I do to systems provides as much help as breaking it into more pieces.

Kent Beck - Smalltalk Best Practice Patterns `97
Why isn’t this common?
Change
Change
Is Hard
Change Is Hard

Even when it's better.
Change Is Hard

Even when it's better.
Demonstrably better.
Change Is Hard

Good code invariably has small methods and small objects. Only by factoring the system into many small pieces of state and function can you hope to satisfy the “once and only once” rule. ... [N]o one thing I do to systems provides as much help as breaking it into more pieces.

Kent Beck - Smalltalk Best Practice Patterns `97
Change Is Hard

Good code invariably has small methods and small objects. Only by factoring the system into many small pieces of state and function can you hope to satisfy the “once and only once” rule.

I get lots of resistance to this idea, especially from experienced developers, but no one thing I do to systems provides as much help as breaking it into more pieces.
Change Is Hard

Biggest Lesson For Me
XP
Is Different
XP
Is Change
XP is Hard

XP requires strong technical practices
XP Is Hard

- Typical coding is imperative

XP requires strong technical practices
XP Is Hard

- Typical coding is imperative
- Good object oriented code interacts differently

XP requires strong technical practices
XP Is Hard

- Typical coding is imperative
- Good object oriented code interacts differently
- We think differently

XP requires strong technical practices
XP Is Hard

- Typical coding is imperative
- Good object oriented code interacts differently
- We think differently

XP requires strong technical practices

Good object oriented code requires a mindset change
XP Is Hard

- Typical coding is imperative
- Good object oriented code interacts differently
- We think differently

Good object oriented code requires a mindset change

- Good practices are change/hard
XP Is Hard

- Typical coding is imperative
- Good object oriented code interacts differently
- We think differently

Good object oriented code requires a mindset change

- Good practices are change/hard
- Easy to be discouraged

XP requires strong technical practices
XP Is Hard

- Typical coding is imperative
- Good object oriented code interacts differently
- We think differently

Good object oriented code requires a mindset change

- Good practices are change/hard
- Easy to be discouraged
- Hard to learn alone

XP requires strong technical practices
XP Works
XP Works, We Know.

It’s why we’re here.
XP Works, We Know.

- Long time w/o substantial adoption

It’s why we’re here.
XP Works, We Know.

- Long time w/o substantial adoption

“I get lots of resistance to this idea, especially from experienced developers”

It’s why we’re here.
XP Works, We Know.

It’s why we’re here.

- Long time w/o substantial adoption

  “I get lots of resistance to this idea, especially from experienced developers”

- Why?
XP Works, We Know.

- Long time w/o substantial adoption

“...I get lots of resistance to this idea, especially from experienced developers”

- Why? - Experience not doing it

It’s why we’re here.
XP Works, We Know.

It’s why we’re here.

• Long time w/o substantial adoption
  “I get lots of resistance to this idea, especially from experienced developers”

• Why? - Experience not doing it
• Some practices showing up
  ○ (Sec)DevOps
  ○ Pair+ Programming
  ○ Collective Code Ownership
Make Change Happen?
Make Change Happen?

- This
Make Change Happen?

- This
- Show the results
Make Change Happen?

- This
- Show the results
- Iron Fist
What Changes?

- How we write the code
What Changes?

- How we write the code
- How we build the code
What Changes?

- How we write the code
- How we build the code
- How we think about code
Absent Technical Practices

- Can’t go faster
- Can’t get ~zero defects
- Can’t test quickly
- Can’t share workload
- Can’t change code easily
With Technical Practices

- 25% the Developer Hours
- Zero Code Defects
Feature Parity in 25% Dev Hours
Technical Practices

- No Getters / No Setters
- `if` Only As Guard Clause
- Isolate Their Code
- Never `null`
- No `new` inline
- Composition, Not Inheritance

- <Strong opinions, loosely held>
- Be Immutable
- No Primitives
- Extract Cohesion
- No Public Statics
- Never Reflection