RESCUING LEGACY SOFTWARE FROM IMPENDING DOOM

martin@nreality.com
@martincronje

jacques@nreality.com
@jacdevos
Your participation...
Why this talk?
Legacy code kills agility
Refactoring is hard
Refactoring stories from the trenches
Why, what, when, not how
Credits

• Kent Beck on Small, Safe Steps and TDD
• Robert C. Martin on Clean Code
• Martin Fowler on Refactoring Techniques
• Ward Cunningham on Technical Debt
• Michael Feathers on Starting with a Mess
• J.B. Rainsberger relating it to Accidental Complication
What is legacy code?

What is the value in fixing legacy code?

How do we speak to Product Owners about it?

Should you rewrite?

What is refactoring?

When should we refactor?
What is legacy code?
Legacy code has a negative perception.
What is legacy code?
You might have said...

- Any code in production
- Ancient code
- Unsupported technology
- Not up to standard
- Code smells
“legacy code is simply code without tests.”

- Michael Feathers
Legacy code is valuable code that we feel afraid to change. - J.B. Rainsberger
Summary: Legacy Code?

Legacy code is risky and expensive to change.
Why fix legacy code?
Why fix legacy code?
You might have said…

• If it ain't broken, don't fix it
• Make it easier to work with
• For our future happiness
• Good craftsmanship
Professionalism
Quality culture
Return on investment
Increasing the future value
In summary

Improve the code to reduce maintenance costs
Do POs understand the value in fixing legacy code?
Do Product Owners understand? What do they say?
You might have said...

- Not enough time
- We cannot afford it
- Business will close-down
- Just one more feature
- They allow us to recover
They think refactoring fails?
They think developers play?
They don’t understand the impact of bad design?
Do they really say that?
How do we explain it to them?
Risk to business continuity
Unpredictable timelines
In summary
Learn how to communicate the impact of legacy code to Product Owners
Should we rewrite?
Should we rewrite?
You might have said...

- No. Rather the devil you know
- No. It’s too expensive
- Yes. Use latest technologies
- Yes. Learnt from mistakes!
Big bang rewrite
Lesson #1: Unpredictable timelines
Lesson #2: Puts your business at risk
Lesson #3:
Always harder than you think
In summary

Resist the temptation to rewrite.
What is the alternative?
Refactoring is a safer bet
...make it easier to understand and cheaper to modify without changing its observable behavior.

- Refactoring, Martin Fowler
Increase ease of change, while decreasing risk of change
Small incremental improvements into production
Cast safety nets as you progress
Start with lowest impact improvement
In summary

Improve design without changing observable behaviour
When do we refactor?
Refactoring workflows

New code | Code changes | Solution wide
Cleaning up as we go
What if this is the campsite?
Handling major problems
Tracer bullet
refactoring
In summary
Clean up as you go, and escalate when there is an impending crisis
How are you going to make it happen?
Code that we’re afraid to change.

Invest in making change easier and less risky.

Talk to Product Owners about return on investment.

Avoid rewrites, favour refactoring.

Many, small improvements without changing behaviour.

Keep it clean and escalate when crisis is imminent.
COMMUNICATE THE VALUE OF IMPROVING THE CODE

martin@nreality.com
@martincronje

jacques@nreality.com
@jacdevos
Join us for this course in:

Jo’burg, South Africa (October)
Oslo, Norway (December)
London, UK (December)

martin@nreality.com
@martincronje

jacques@nreality.com
@jacdevos
COMMUNICATE THE VALUE OF IMPROVING THE CODE

martin@nreality.com
@martincronje

jacques@nreality.com
@jacdevos
Reference

• Fowler, Martin (1999). Refactoring: Improving the design of existing code. Addison Wesley.
• Rainsberger, J.B. Fundamental Theorem of Agile Software Development https://vimeo.com/79106557