How to deal with
Progress Legacy Code
TRANSLVANIA
Agenda

1. What
2. It
3. Face it! Progress it!
Do you remember the first line of code you wrote?
WHAT?

Code we've gotten from someone else.
Code we have to change but don't understand.
Fragile code.
(Pre)Historical code, probably only you care about.
Characteristics

Old
Rigid
Fragile
Huge
Unreadable
Examples

1990

If Customer.age > 18 then
display Customer.name.

2017

If Customer.age > 18 then
display Customer.name.
else
  if Customer.age < 30 then
    Customer.deadline = 2018.
  else
    if Customer.age < 50 and
    Customer.age > 45 then
      display "win-win".
    else
      if Customer.age <> 28 then
        display Customer.deadline.
In another words...

Poor architecture
Non-uniform coding styles
Poor written documentation
"Oral" documentation lost
No tests

But...
Valuable!

...because only successful code becomes legacy
Why do we have legacy code?

More & more features
Shortcuts, Hacks & Workarounds - Our best friend
Developer rotation & the Developer's team grows
Less communication

Fact:
Programmers usually find it mentally less taxing to place a new feature in an existing class than to create a new class for the feature.
Why we change legacy code?

- We need new features
- We have to fix it
- Refactoring
- Optimization

Why we do NOT want to change legacy code?

- It looks ugly
- It works
- Not easily fixed
- Fear
Possible obstacles
But it's in my way...

I don't have time

I don't understand it

Unstructured code

Breaking dependencies
It doesn't work...... why?

It works...... why?

9GAG is your best source of fun.
public static void Main()
{
    labelA;
    if ( _ )
        goto labelC;
    if ( _ )
        goto labelB;

    labelD;
    if ( _ )
        goto labelE;
    labelC

    labelE;
    if ( _ )
        goto labelA;
    if ( _ )
        goto labelD;
    labelB;
}
Use techniques

Avoid temptation!
Go step-by-step

Design improvements,
not the main goal

Purpose:
Make code TESTABLE
Face it!
Fight it!
Start from the beginning...

And maybe you'll finish one day...

Or...

Refactor
Risks

Changing UC often feels like trying to find our way out of a labyrinth...

You'll have to go back a few times... And try again!

"OK, I think we're pretty much back to square one. You can stop now."

Trial & Error process. Make educated guesses
You’ll gave to go back a few times.. And try again!

"OK, I think we’re pretty much back to square one. You can stop now."
Trial & Error process.
Make educated guesses
Changing LC often feels like trying to find our way out of a labyrinth...
Changing legacy code

Current situation

You are afraid to change the system
The LC change algorithm

1. Identify change point
2. Break dependencies
3. Find test points
4. Write tests
5. Make changes and refactor
The monster we may find...

- The same code more than once
- A single class, doing too much
- Technical debt
- Code cancer
- Code does not make any sense
- Is this my code?
- Low quality

Fear no evil!
Use the boy scout rule
Prevent it!

Identify requirements → Understand the code → Modify

Identify requirements → Understand the code → Modify

WRITE TESTS → Modify
Testing Legacy Code

- The code is stable
- Writing tests is not as easy as you thought
- Write test only for the code you've changed
- Never change without a purpose
Tests...

Maintainable
Should have minimal duplication
Avoid testing multiple things in one test

Trustworthy
Does not repeat the logic from the code
Repeatable 100% of the time
Avoid using the real DB or real file systems

Readable
Follow A.A.A as much as possible
Use naming conventions
Maintain visibility of values
How?

Find the right ATTITUDE
- Always leave the code behind in a better state than you found it.
- You have to care
- Don’t give up
- Small victories
- Discipline is the key

Find the best PRACTICES
- Cut the bad parts out, do not comment them
- Tests are your safety net for not creating unintentional changes
- Refactoring should be an ongoing process that is part of your daily work.
- Use strict Coding Guidelines
- Document your work
- Reducing complexity (limit the pieces of code that are dependent on other pieces of code)
Progress It!

Fight legacy code!
Static code analysis

**SonarQube**
- Free & Open source "Code Quality Platform" - plugin for OpenEdge
- Provides quality snapshots on build level
- Tracks developers' "sins"
- Support for various programming languages via plug-ins

**Prolint**
- Tool for automated source code review of Progress 4GL code.
- It examines the source files for bad programming practices.
- Works with a customizable library of "rules".

**Proparse**
- Open source ABL parser
- Dynamic linked library which parses Progress 4GL source code
Why?

No improvement without measurements
We maintain "legacy"
The program is never "finished"
Complexity creeps in
Broken windows theory (bad designs, wrong decisions, or poor code)
Software entropy (the amount of disorder)
Code rots

How?

Locates problems or potential bugs
Violation of coding-standards
Code duplication
Unit-Test coverage
Web-Dashboard
Reporting Utility
Eclipse Integration
## Unit testing frameworks

Your code is broken, you just don’t know it yet.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Features</th>
</tr>
</thead>
</table>
| **ProUnit** | - test cases are Progress ABL files that follow simple rules  
- you can save results in many forms (XML, CSV etc.)  
- supports only procedures |
| **OEUnit** | - test cases are classes containing test methods  
- can be embedded  
- supports only classes |
| **ABLUnit** | - simple and powerful framework  
- integrated since OpenEdge 11.4 (no installation required)  
- similar with OEUnit  
- supports both classes and procedures |
Pros

- problems are found earlier
- you can easily locate bugs
- makes the process Agile - allows to make changes more easily
- improves the quality of code
- writing tests forces you to think through your design
- reduce costs
- the tests are a form of self documentation

Cons

- writing tests is time consuming
Don't let Legacy Code always win over Good Code!

Ask yourself: Does Legacy Code Writer really exist?

Do not ever copy-paste!

Keep in mind that two bad choices don't compensate, they accumulate!

Miss a unit test, and be sure that the next team will miss them all.

Every time you don't code as a perfectionist, you are condemning the future programmer (and the future yourself) to under-perform.

Don't leave "broken windows!"

PS: Everyone can write code from scratch, and sustain it for the first 3 months. BUT only professional programmers can maintain it for the next years.
How to deal with
Progress Legacy Code