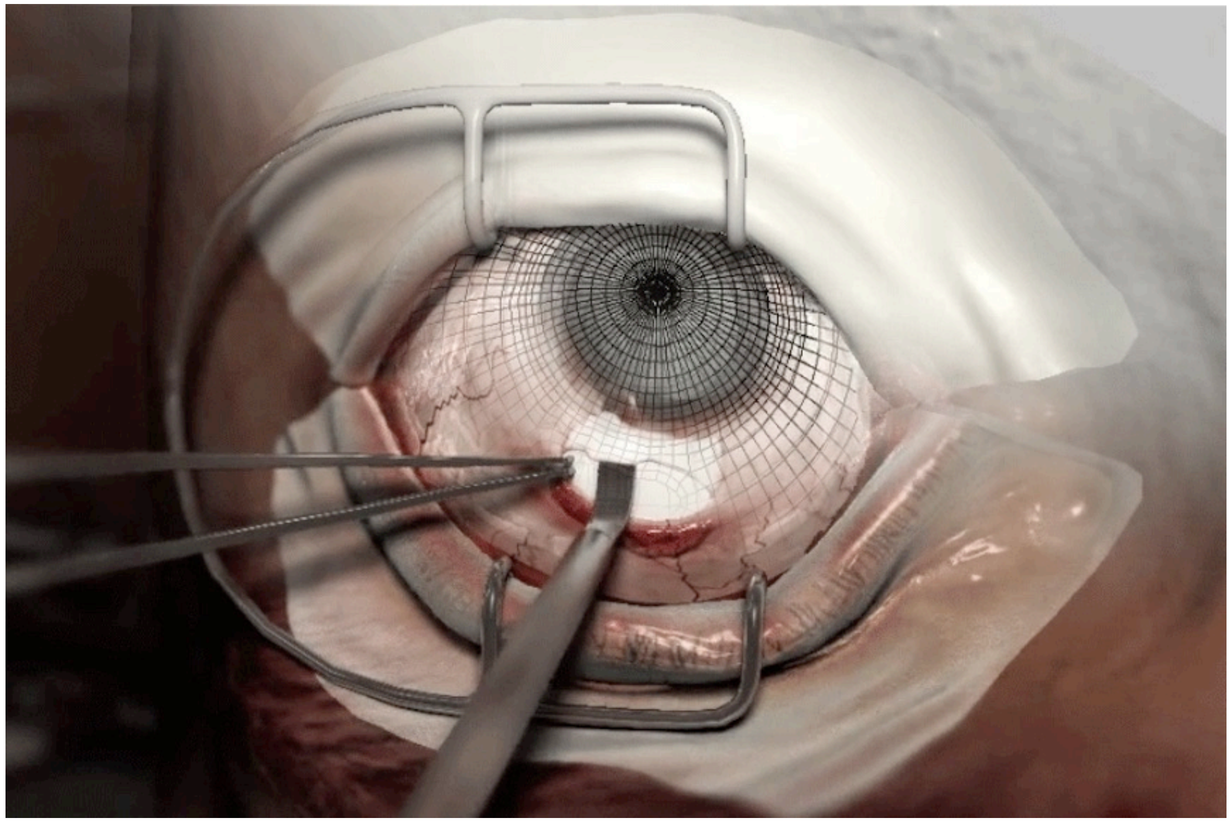
SOFTWARE MAINTENANCE

Jeremie Dequidt

INTRODUCTION

- Assistant Professor in Computer Science
- Research activities: interactive simulation, computer graphics, virtual / augmented reality
- Applications in medicine and soft robotics



INTRODUCTION

- Development of SOFA (https://www.sofa-framework.org)
- Since 2006
- GitHub stats: 249 ⑂ 606 ☆ 93 👥
- 2 releases / year -> Linux, Win*, MacOS
- 850k loc, 60+ plugins (10m loc)
- 4 transfers of technology, 1 international patent

NOTES

- These slides have been largely influenced by Nicolas Anquetil, Benoît Combemale courses
- … and Anne Etien

PURPOSES OF THE COURSE

- Understanding the importance of maintenance
- Knowing the test mechanism
- Developping tests first
- Better understanding the object paradigm
- Knowing the foundations of software quality
- Knowing the rudiments of visualization
- Discovering the continuous integration principles
- Studying the quality of unknown software
- Enhancing your own development.

COURSE ORGANISATION

- Introduction to maintenance
- Test driven development
  - Practice 2 hours
- Continuous Integration, Clean Code
  - Practice 3x 2 hours

COURSE EVALUATION

- Project restitution (gitlab repository)
- Exam

GOALS

- Why this course?
  - Soft.Maint. is important
  - Soft.Maint. is poorly understood
  - Soft.Maint. is poorly performed

GOALS

- You understand
  - Why software maintenance exists
  - Why you did not like it
  - Why you should like it
  - Know some good practices

AGENDA

- Introduction (definitions)
- Importance of the topic
- Some facts
- Consequences

DEFINITION

> *Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes.*
>
> *ISO/IEC 14764:2006 Software Engineering —*
> *Software Life Cycle Processes — Maintenance*

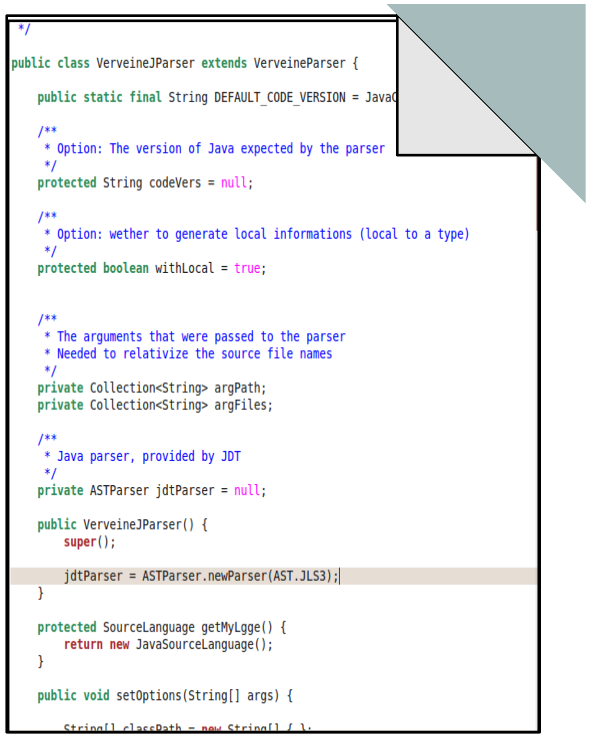
DEFINITION

> *Legacy software: A system which continues to be used because of the cost of replacing or redesigning it and often despite its poor competitiveness and compatibility with modern equivalents. The implication is that the system is large, monolithic and difficult to modify.*
>
> *mondofacto.com/facts/dictionary*

AGENDA

- Introduction (definitions)
- Importance of the topic
- Some facts
- Consequences

LEGACY SOFTWARE

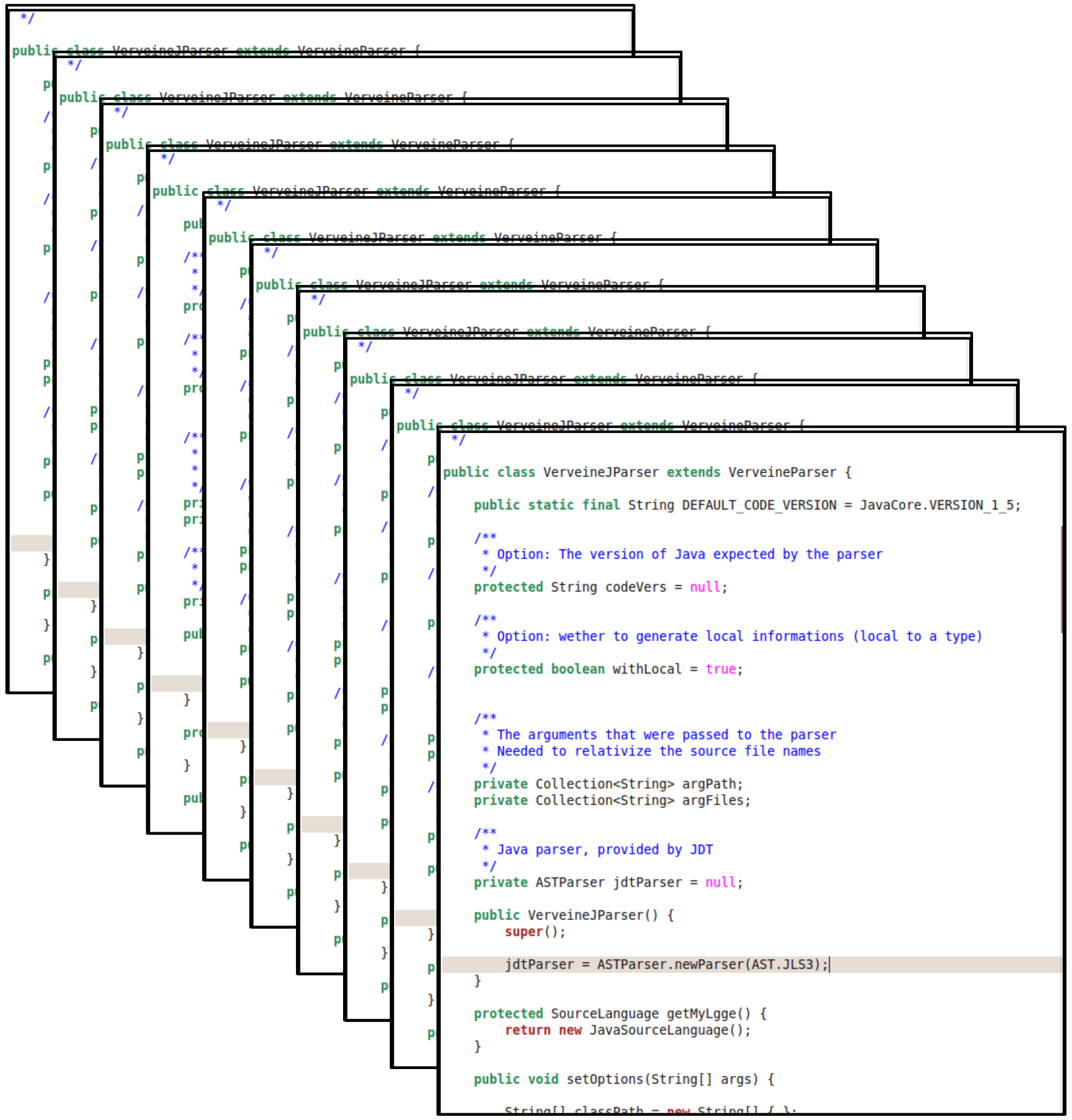


```
 */

public class VerveineJParser extends VerveineParser {

    public static final String DEFAULT_CODE_VERSION = JavaC

    /**
     * Option: The version of Java expected by the parser
     */
    protected String codeVers = null;

    /**
     * Option: wether to generate local informations (local to a type)
     */
    protected boolean withLocal = true;


    /**
     * The arguments that were passed to the parser
     * Needed to relativize the source file names
     */
    private Collection<String> argPath;
    private Collection<String> argFiles;

    /**
     * Java parser, provided by JDT
     */
    private ASTParser jdtParser = null;

    public VerveineJParser() {
        super();

        jdtParser = ASTParser.newParser(AST.JLS3);
    }

    protected SourceLanguage getMyLgge() {
        return new JavaSourceLanguage();
    }

    public void setOptions(String[] args) {

        String[] classPath = new String[] { };
```

1 sheet ≃ 60 lines of code (LOC)

both sides = 120 LOC

LEGACY SOFTWARE



```
 */
public class VerveineJParser extends VerveineParser {

    public static final String DEFAULT_CODE_VERSION = JavaCore.VERSION_1_5;

    /**
     * Option: The version of Java expected by the parser
     */
    protected String codeVers = null;

    /**
     * Option: wether to generate local informations (local to a type)
     */
    protected boolean withLocal = true;


    /**
     * The arguments that were passed to the parser
     * Needed to relativize the source file names
     */
    private Collection<String> argPath;
    private Collection<String> argFiles;

    /**
     * Java parser, provided by JDT
     */
    private ASTParser jdtParser = null;

    public VerveineJParser() {
        super();

        jdtParser = ASTParser.newParser(AST.JLS3);
    }

    protected SourceLanguage getMyLgge() {
        return new JavaSourceLanguage();
    }

    public void setOptions(String[] args) {

        String[] classPath = new String[] { };
```

10 sheets = 1200 LOC

(1.2 KLOC)

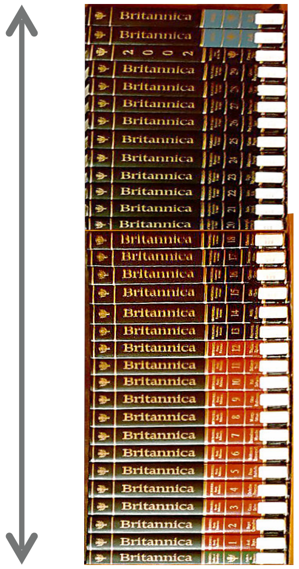
LEGACY SOFTWARE

Windows NT 3.1 (1993)

- 4 to 5 MLOC

3.75 m

3.20 m



Encyclopedia Britanica (15 ed., 32 volumes)

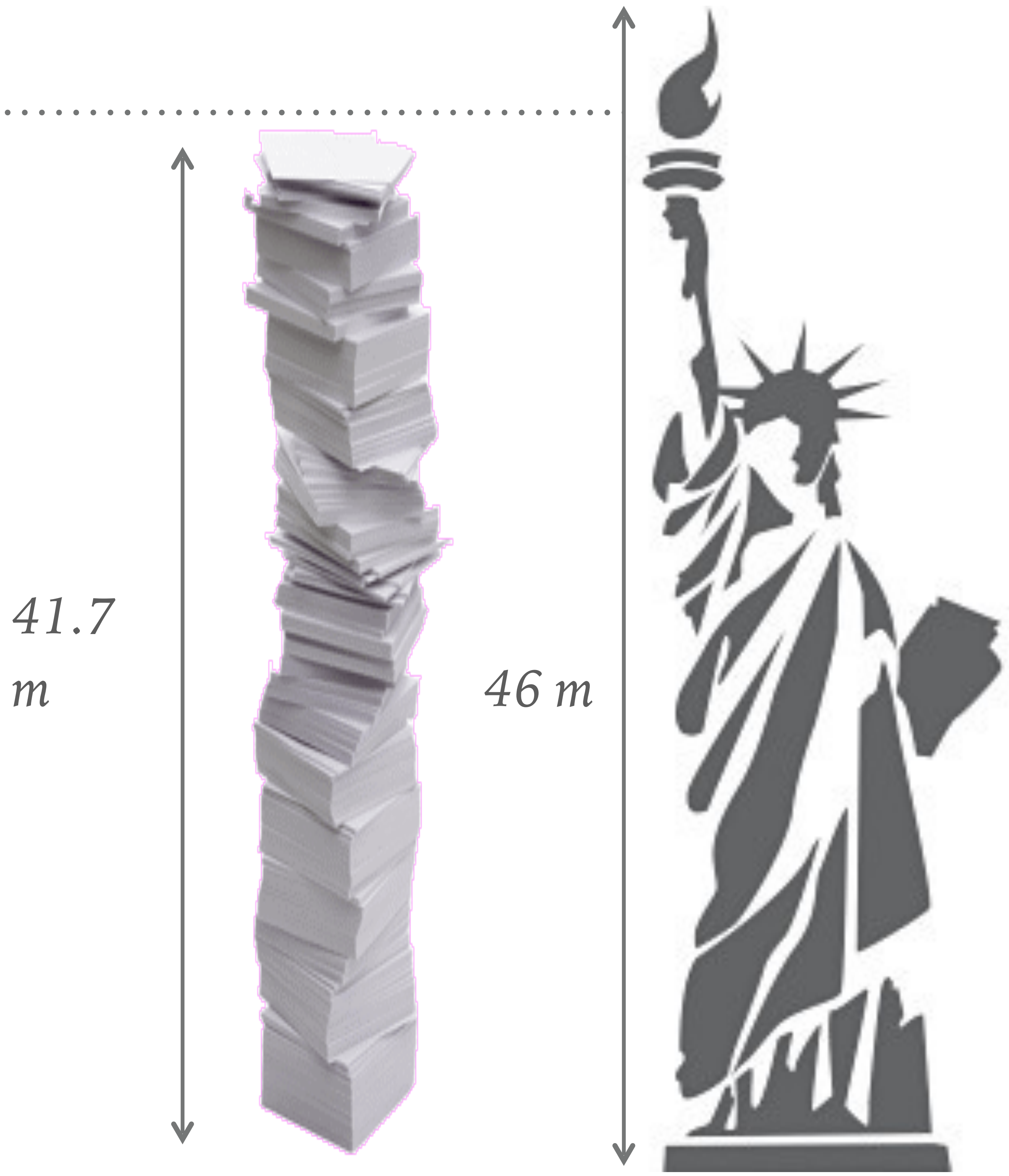
LEGACY SOFTWARE

Windows NT 3.1 (1993)

4 to 5 MLOC

Windows server 2003

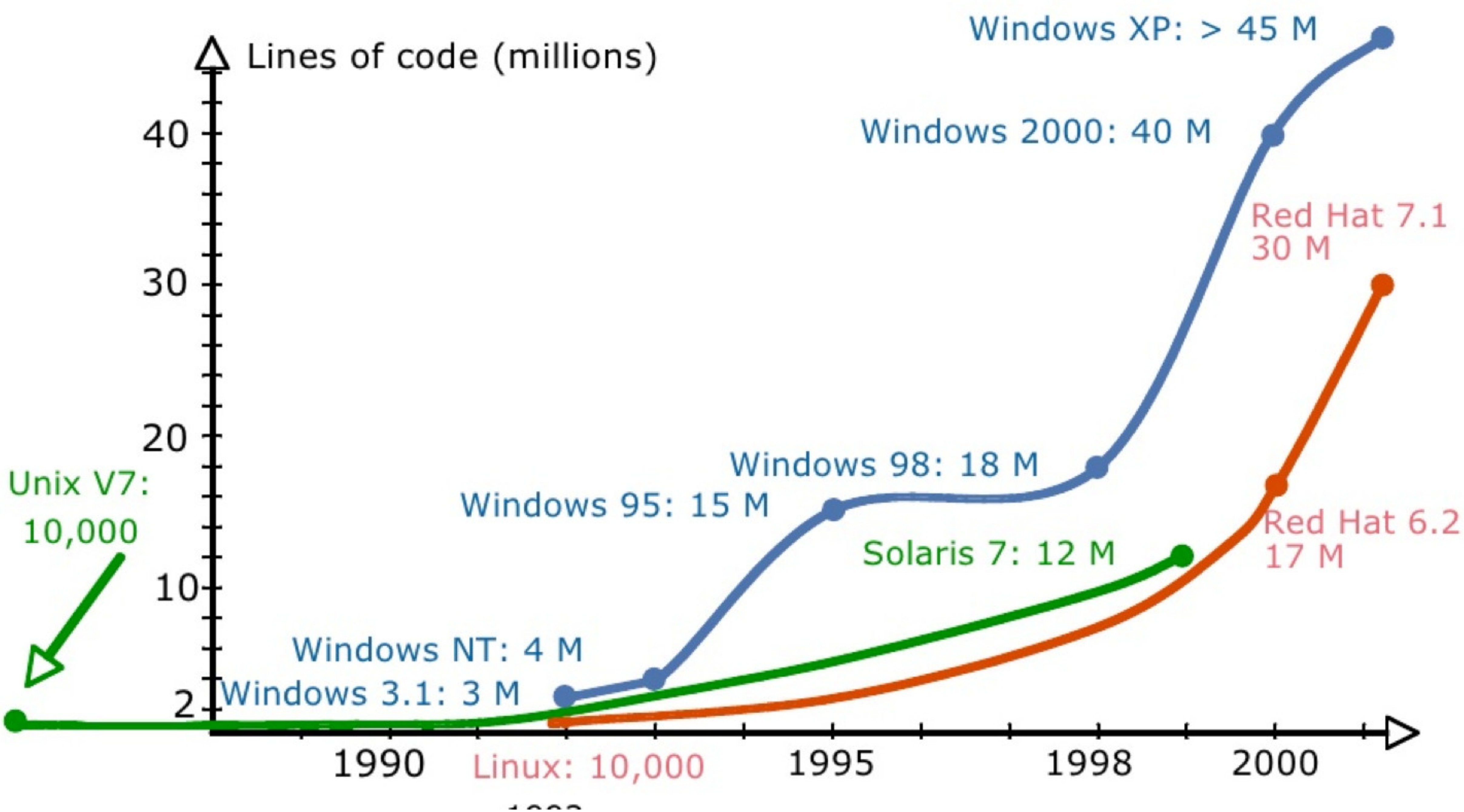
50 MLOC



LEGACY SOFTWARE

- Linux kernel 3.6
  → 16 MLOC
- MacOS X 10.4
  → 86 MLOC
- Debian 5.0
  → 324 MLOC

LEGACY SOFTWARE



RELEVANCE?

- Estimations:
  - 120 billion LOC maintained in 1990 (Ulrich, 1990)
  - 200 billion in 2000 (Sommerville, 2000)

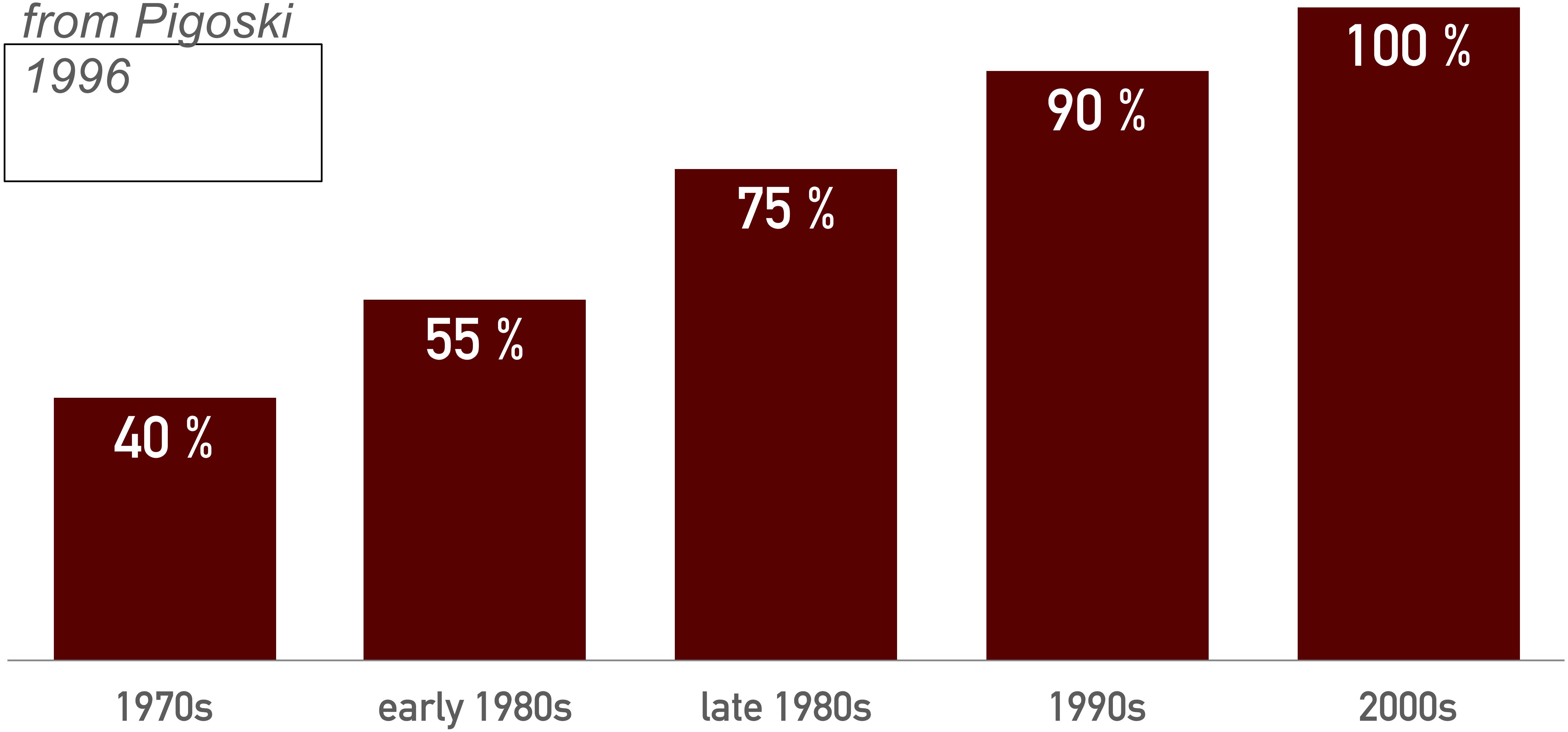
RELEVANCE?

- Annual cost in USA > $70 billion (Sutherland, 1995; Edelstein, 1993)
- Nokia spent $90 million on Y2K
- US government spent > $8 billion

RELEVANCE?

Cost of maintenance in a software life

from Pigoski 1996



AGENDA

- Introduction (definitions)
- Importance of the topic
- Some facts
- Consequences

SOME FACTS

- Dominant activity in software engineering
- Yet, still poorly understood and despised
  - Punishment
  - Probation
  - No career advancement

TRUE/FALSE?

- Maintenance can be eliminated with perfect developmen
- Maintenance will be solved by modern technology (ex. Model Driven Development)
- Maintenance is difficult and boring
- Better restart from scratch

True	False

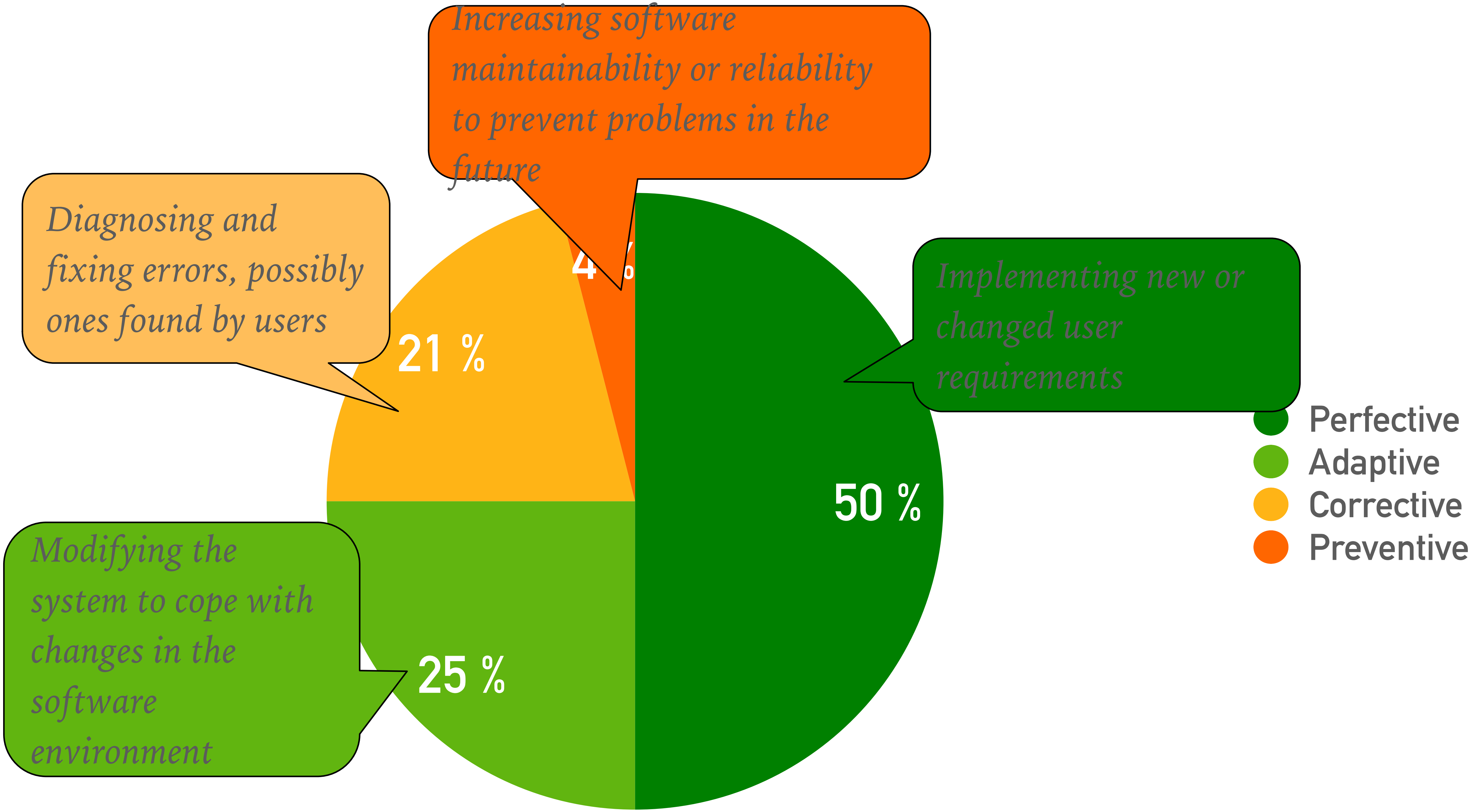
ELIMINATE MAINTENANCE?

- Why didn't they do it right in the first place?!?
- I am loosing time correcting other peoples' mistakes!

ELIMINATE MAINTENANCE?

- Development techniques improve all the time
  - Software processes (Agile, TDD)
  - Software quality (CMMI)
  - Tools (IDEs, xUnit)
  - Languages (AOP, MDD)
- Maintenance problem still exist !

MAINTENANCE CATEGORIES



Increasing software maintainability or reliability to prevent problems in the future

Diagnosing and fixing errors, possibly ones found by users

Implementing new or changed user requirements

Modifying the system to cope with changes in the software environment

HARDWARE / SOFTWARE

- Hardware maintenance:
  - replacement of used parts
- Software maintenance:
  - Source code doesn't wear
  - Maintenance is mainly evolution
  - Little bug correction

> *Software systems must be continually adapted or they become progressively less satisfactory*
>
> *First law of software evolution [Lehman, 1974]*

SOFTWARE AND ENVIRONMENT

- A system works within the real world
- The world changes:
  - New business opportunities
  - Growing user expectations
  - New laws …
- **Software systems must evolve or die (not useful)**
- **Maintenance is mainly due to external causes**

THE MUSSEL SHACK

- Once upon a time, a fisherman in Dunkerque opened a small mussel selling point

THE MUSSEL SHACK

- Business was good

THE MUSSEL SHACK

- Business was very good

THE MUSSEL SHACK

- Employees asked for a cafeteria

THOMAS OYSTER CO
THOMAS OYSTER CO
THOMAS OYSTER CO
THOMAS OYSTER CO
Quick

THE MUSSEL SHACK

- Directors requested their dinning room

THE MUSSEL SHACK

- Law imposed an emergency exit

THOMAS OYSTER CO
THOMAS OYSTER CO
THOMAS OYSTER CO
THOMAS OYSTER CO
Qui

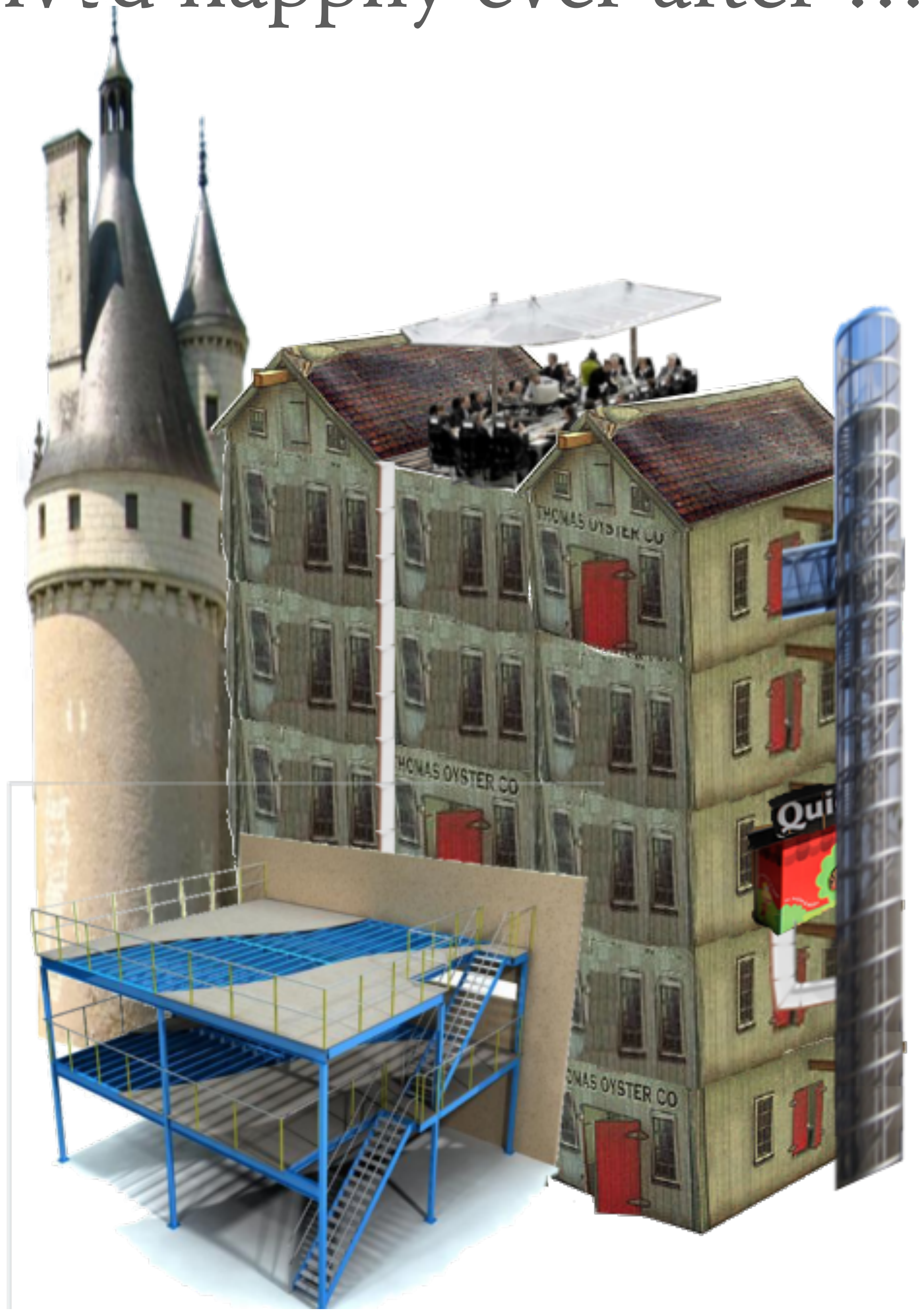
THE MUSSEL SHACK

- Concurrents have fitness room, added a piscine

THOMAS OYSTER CO
THOMAS OYSTER CO
THOMAS OYSTER CO
THOMAS OYSTER CO
Qui

THE MUSSEL SHACK

- and they lived happily ever after …



MORAL

Maintenance is a sign of success! The system is used and useful,the users want more

MORAL

Mainten

Well developed systems will receive more maintenance

MORAL

To eliminate maintenance, create bad systems

→ few users

→ too difficult to modify

will

rece

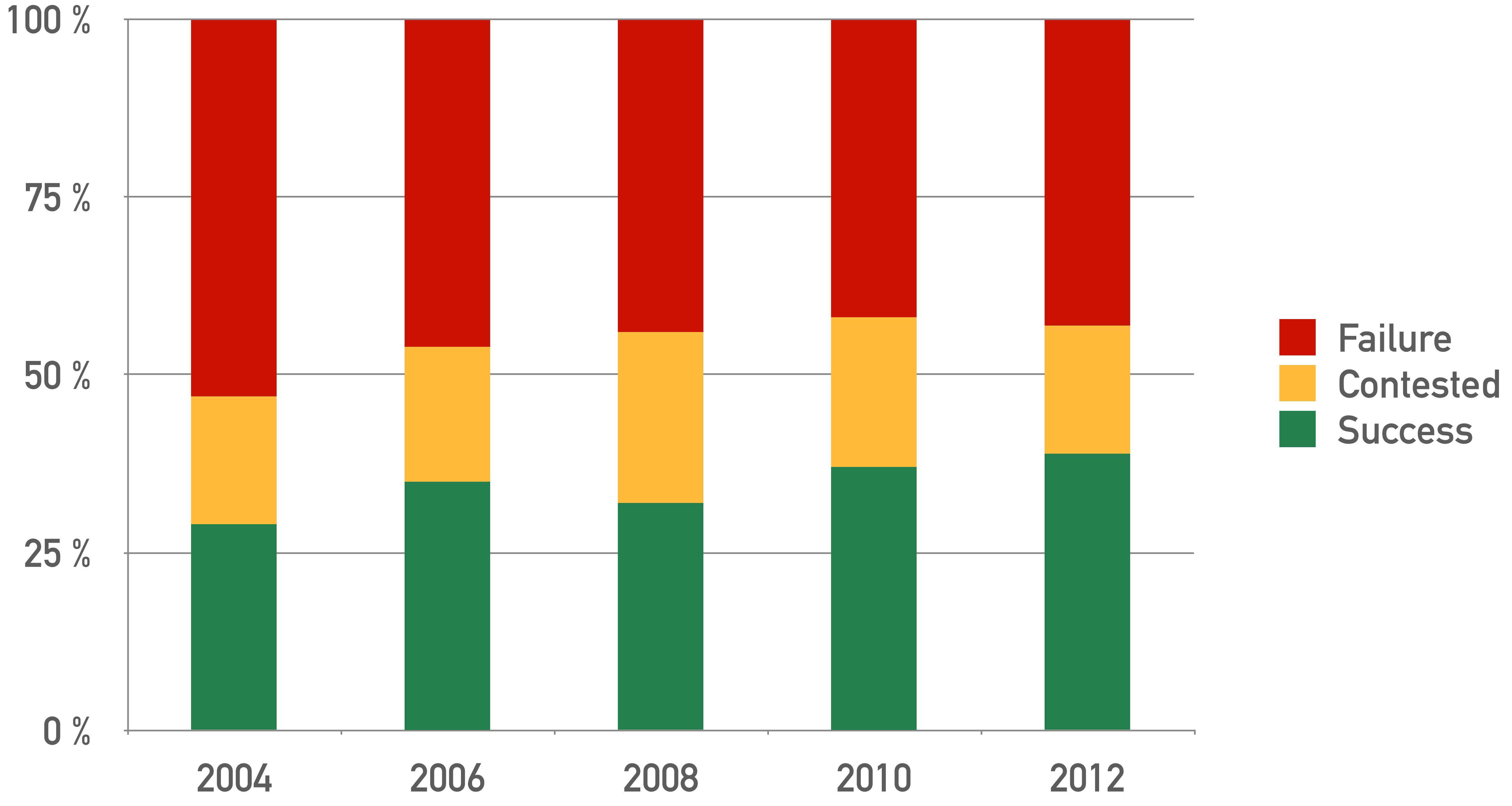
MORAL

The better the system, the more maintenance (evolution) it will require !

TRUE/FALSE?

	True	False
➤ Maintenance can be eliminated with perfect developme		✓
➤ Maintenance will be solved by modern technology (ex. Model Driven Development)		
➤ Maintenance is difficult and boring		
➤ Better restart from scratch		

STANDISH GROUP STUDY ON THE SUCCESS OF SOFTWARE PROJECTS



LONG TERM AVAILABILITY

- Airbus A300 Life cycle
  - Program began in 1972, production stopped in 2007
    - 2007-1972 = **35 years**
  - Support will last until 2050
    - 2050-1972 = **78 years!!**

NEW TECHNIQUES

- Cobol > 60% of all code in the world [eWeeks, 2001]
- 180 GLOC in use, + 1GLOC/year [Gartner, 2006]

NEW TECHNIQUES

- ➤ Cobol – 1959

LIFE
CRASH DIETS ARE NO GOOD
TURMOIL ON CAPITOL HILL: A LOOK AT A NEW KIND OF CONGRESS
CASTRO IN TRIUMPHANT ADVANCE TO HAVANA
2NeatBooks.com
JANUARY 19 1959
25 CENTS

25 septembre 1959 - Charles DE GAULLE, au 6 d'HALLICOURT

NEW TECHNIQUES

- Ada – 1983
  - Creation of Internet (562 hosts)
  - Macintosh did not exist
  - MS Windows was announced (v1.0 in 1985)

NEW TECHNIQUES

- Ada – 1983

NEW TECHNIQUES

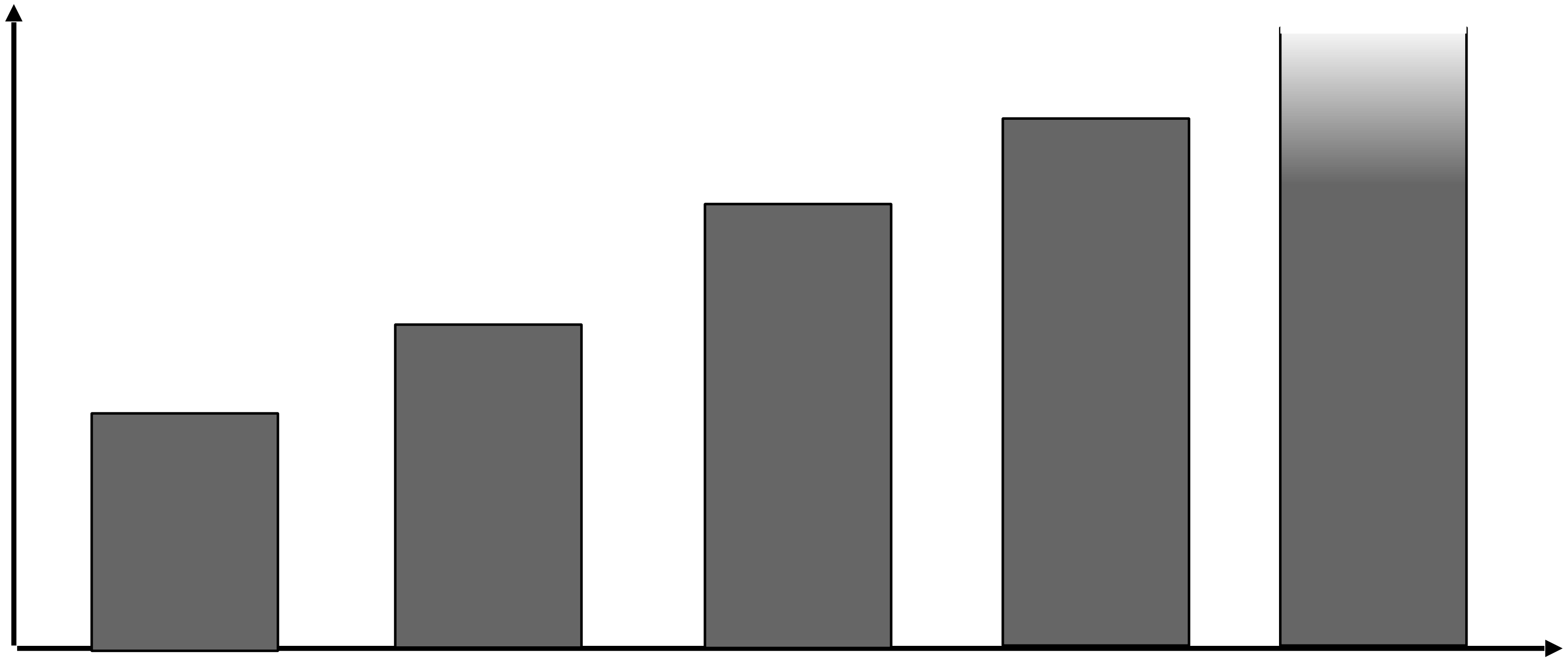
- New techniques do not target:
  - Past technologies (Ada, Cobol)
  - Existing systems

TECHNIQUES IMPROVEMENT

- Development techniques improve all the time
  - Software processes (Agile, TDD)
  - Software quality (CMMI)
  - Tools (IDEs, xUnit)
  - Languages (AOP, MDD)
- Maintenance problem still exist!

NEW TECHNIQUES

- New techniques do not target existing legacy software
- Miss 90+ % of the market



NEW TECHNIQUES

- New techniques (models) are still programs
- Programs are models of the world
- They will need to be maintained

TRUE/FALSE?

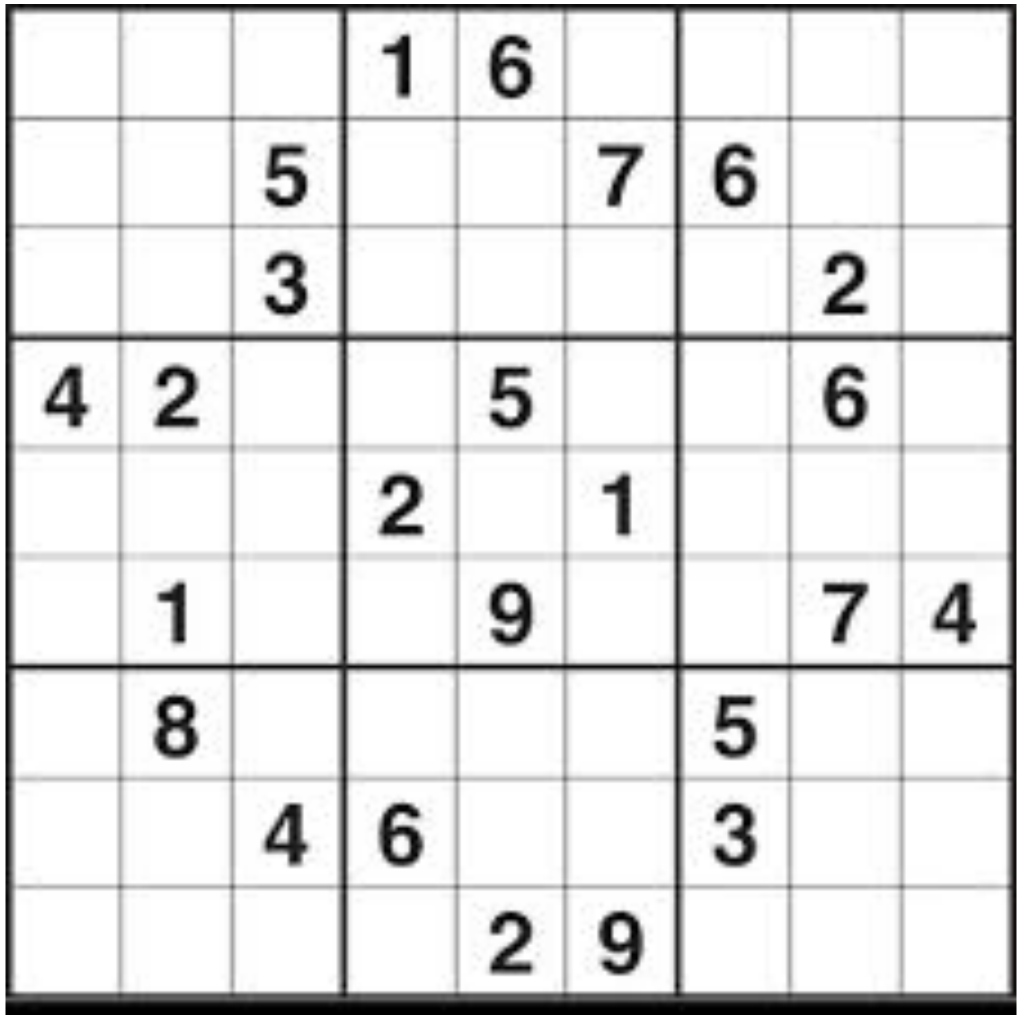
	True	False
➤ Maintenance can be eliminated with perfect developme		✓
➤ Maintenance will be solved by modern technology (ex. Model Driven Development)		✓
➤ Maintenance is difficult and boring		
➤ Better restart from scratch		

MORE DIFFICULT?

- Intrinsically more difficult than development
    - Information missing on existing system
    - Must preserve some backward compatibility (existing data, user habits, …)
    - More chaotic (reaction to external events)
    - Less resources
    - …

BORING?

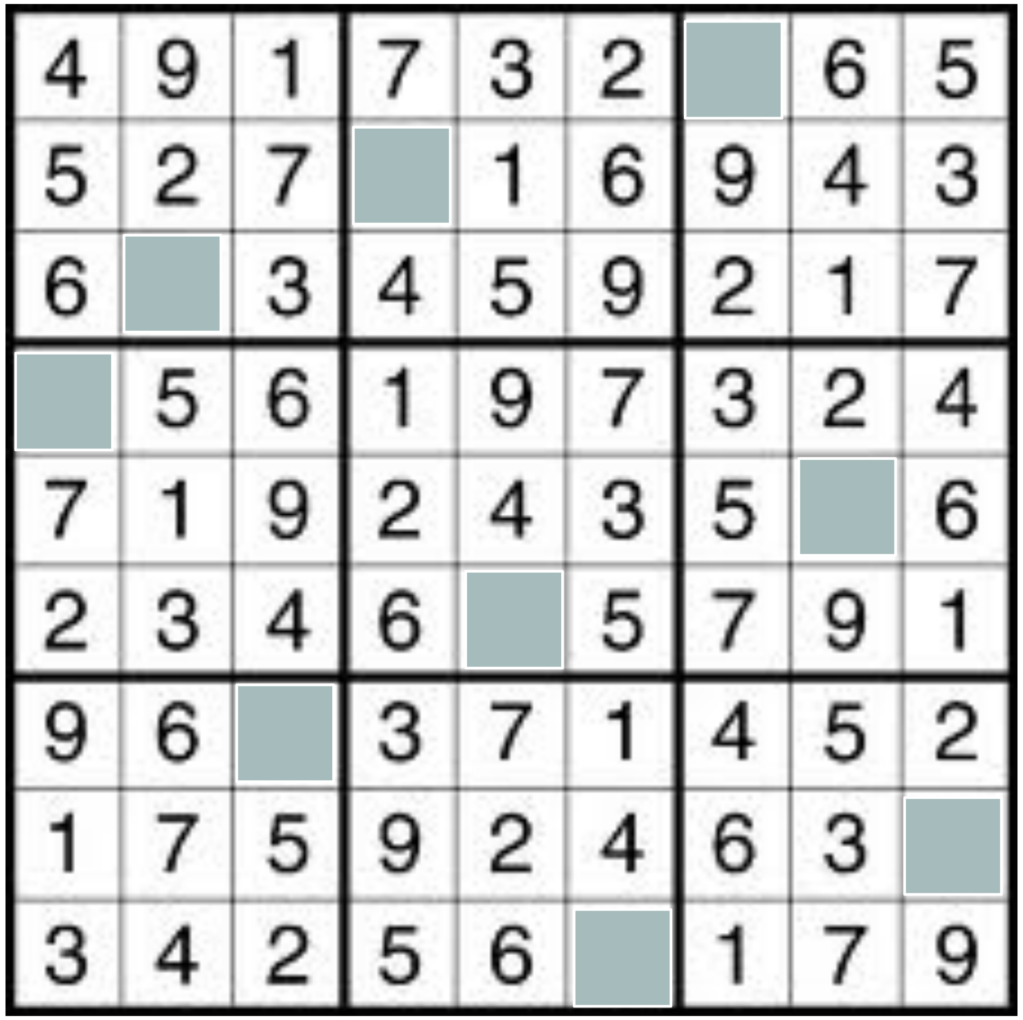
- Difficult ≠ Boring



			1	6				
		5			7	6		
		3					2	
4	2			5			6	
			2		1			
	1			9			7	4
	8					5		
		4	6			3		
				2	9			

BORING?

- Easy ≠ Interesting



4	9	1	7	3	2		6	5
5	2	7		1	6	9	4	3
6		3	4	5	9	2	1	7
	5	6	1	9	7	3	2	4
7	1	9	2	4	3	5		6
2	3	4	6		5	7	9	1
9	6		3	7	1	4	5	2
1	7	5	9	2	4	6	3	
3	4	2	5	6		1	7	9

BORING?

- Maintenance is difficult
- Can be seen as an interesting challenge
- A good way to learn many things (e.g. programming tricks)

TRUE/FALSE?

	True	False
➤ Maintenance can be eliminated with perfect developme		✓
➤ Maintenance will be solved by modern technology (ex. Model Driven Development)		✓
➤ Maintenance is difficult and boring	✓	✓
➤ Better restart from scratch		

RESTART FROM SCRATCH

- Intuitively obvious solution

RESTART FROM SCRATCH

- Software ≠ Hardware
- Legacy software is successful
- New software =
  - Costs
  - New bugs
  - Teaching user new habits
- Experience shows it can go very wrong

LOUVOIS EXAMPLE

- Errors of payment computation in 2012: 465 millions euros
  - Hundreds of militaries have not been paid during several months.

- *In 1996, the French Army ministry launched a project to unify the payment inter-armies.*
- *After several failures, the project entered in production in April 2011.*
- *It was abandoned in 2013*
- *Global cost of the project: 80 million euros*

RESTART FROM SCRATCH

- Recommended action: re-engineer
  - Less risky
  - Iterative approach
  - Build on tested and proved solution
- Down side
  - Future constrained by the past

TRUE/FALSE?

	True	False
➤ Maintenance can be eliminated with perfect developmen		✓
➤ Maintenance will be solved by modern technology (ex. Model Driven Development)		✓
➤ Maintenance is difficult and boring	✓	✓
➤ Better restart from scratch		✓

AGENDA

- Introduction (definitions)
- Importance of the topic
- Some facts
- Consequences

CONSEQUENCES

- The problem is cultural first
- Maintenance is not taught (implies it is not important?)
- Computer science evolves fast (“newer is better”)
- Technology evolves fast (Cobol, Ada on iPhone?)

CONSEQUENCES

- First need to change perception

> *Legacy software: A system which continues to be used because of the cost of replacing or redesigning it and often despite its poor competitiveness and compatibility with modern equivalents. The implication is that the system is large, monolithic and difficult to modify.*
>
> *mondofacto.com/facts/dictionary*

CONSEQUENCES

- First need to change perception

Legacy software: A system which c　*s to*
be used because of the c
redesigning it an i
compet　*rn*
equiva　*n*
is large,　*y.*

mondofacto　/dictionary

> *"Legacy code" often differs from its suggested alternative by actually working and scaling.*
>
> Bjarne Stroustrup

CULTURAL PROBLEM

- Wrong ideas about it
- → Prejudice against it
- → Not studied
- → Not understood
- → Wrong ideas about it

CULTURAL PROBLEM

> *Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes.*
>
> *ISO/IEC 14764:2006 Software Engineering —*
> *Software Life Cycle Processes — Maintenance*

CULTURAL PROBLEM

Software maintenance is the modification of a software pr uct after delivery to correct f rove performance es.

This is a mistake

ISO/IEC 14764:2006 Software Engineering — Software Life Cycle Processes — Maintenance

MAINTENANCE SHOULD BE PREPARED

- Start before delivery
  - Who will maintain?
  - What technology do they know?
  - How to pass knowledge to them?
- Note: Maintenance is a knowledge intensive activity
  - 40% to 60% of the time is spent on studying the system
- Processes are different
  - Maintenance involves a much longer analysis activity
  - Maintenance less planned, more chaotic (external events)
  - Requires a different approach

CLOSING REMARKS

- Software evolution is very important
- Need to change the habits
- Need to invest in maintenance
  - Tools
  - Training

SOME VIDEOS TO GO FURTHER

- https://www.youtube.com/watch?v=i8J20IjuwTw in French (1h20)