Software Re-Engineering

Greenfield Software Development

Greenfield Software Development

Refers to developing a system for a totally new environment and requires development from a clean slate – no legacy code around.

Used when you’re starting fresh and with no restrictions or dependencies.

A pure Greenfield project is quite rare these days, you frequently end up interacting or updating some amount of existing code or enabling integrations.

Examples of Greenfield software development include:

- setting up a new data center,
- or even implementing a new rules engine.
- Building a website or application from scratch.

Advantages - Greenfield Software Development

•Gives an opportunity to implement a state-of-the-art technology solution from scratch.

•Provides a clean slate for software development

•No compulsion to work within the constraints of existing systems or infrastructure

•No dependencies or ties to existing software, preconceived notions, or existing business processes

Disadvantages - Greenfield Software Development

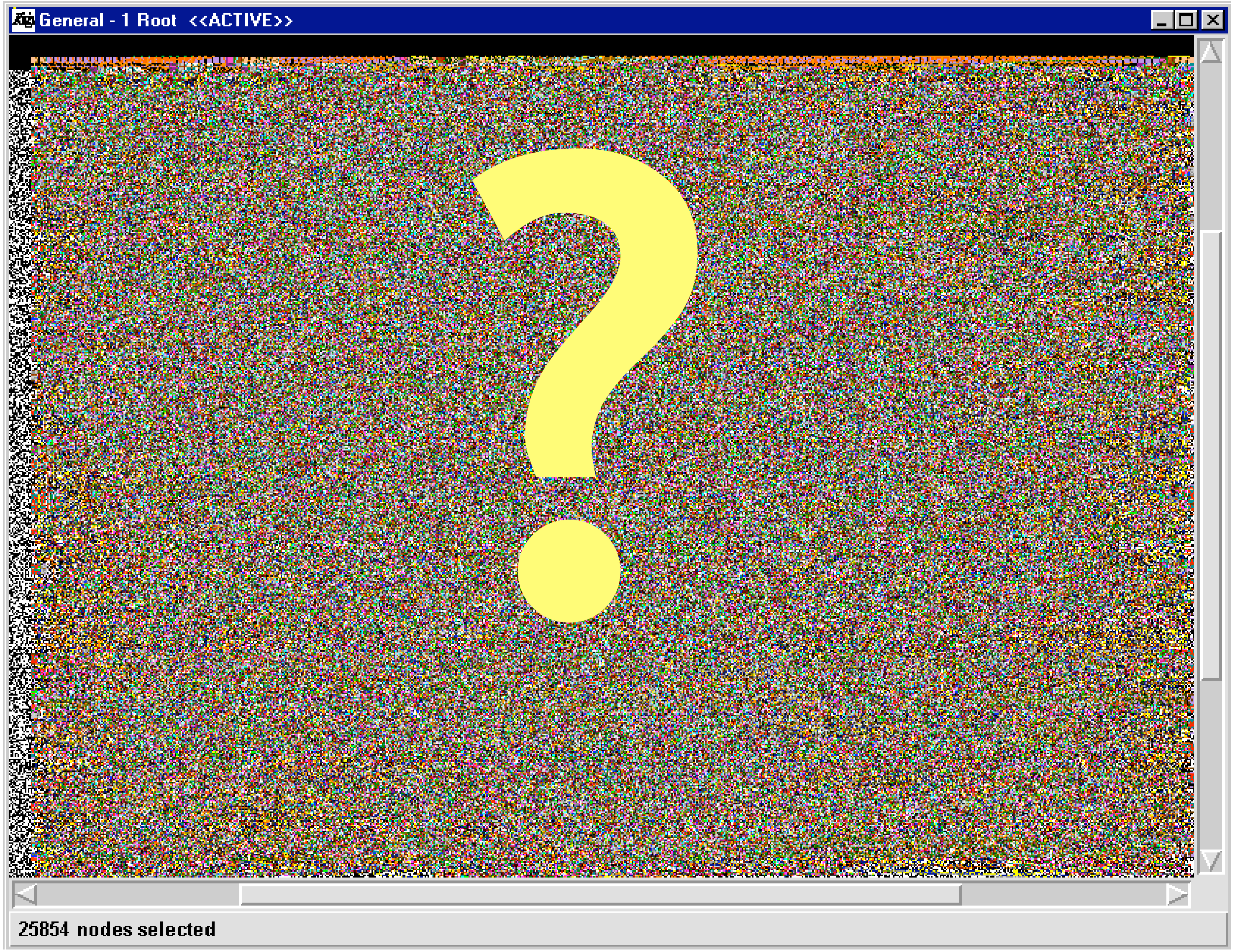
•With no clear direction, the degree of risk is comparatively higher

•Since all aspects of the new system need to be defined, it can be quite time consuming

•With so many possible development options, there may be no clear understanding of the approach to take

•It may be hard to get everyone involved to make critical decisions in a decent time frame

Brownfield Software Development



Brownfield Software Development

Refers to the development and deployment of a new software system in the presence of existing or legacy software system.

Usually happens when you want to develop or improve upon an existing application, and compels you to work with previously created code.

Therefore, any new software architecture must consider and coexist with systems already in place

Examples include:

- adding a new module to an existing enterprise system,
- integrating a new feature to software that was developed earlier,
- or upgrading code to enhance the functionality of an app.

Advantages - Brownfield Software Development

•Offers a place to start with a predetermined direction

•Gives a chance to add improvements to existing technology solutions

•Supports working with defined business processes and technology solutions

•Allows existing code to be reused to add new features

Disadvantages - Brownfield Software Development

•Requires thorough knowledge of existing systems, services, and data on which the new system needs to be built

•There may be a need to re-engineer a large portion of the existing complex environment so that they make operational sense to the new business requirement

•Dealing with legacy code slow down the development process.

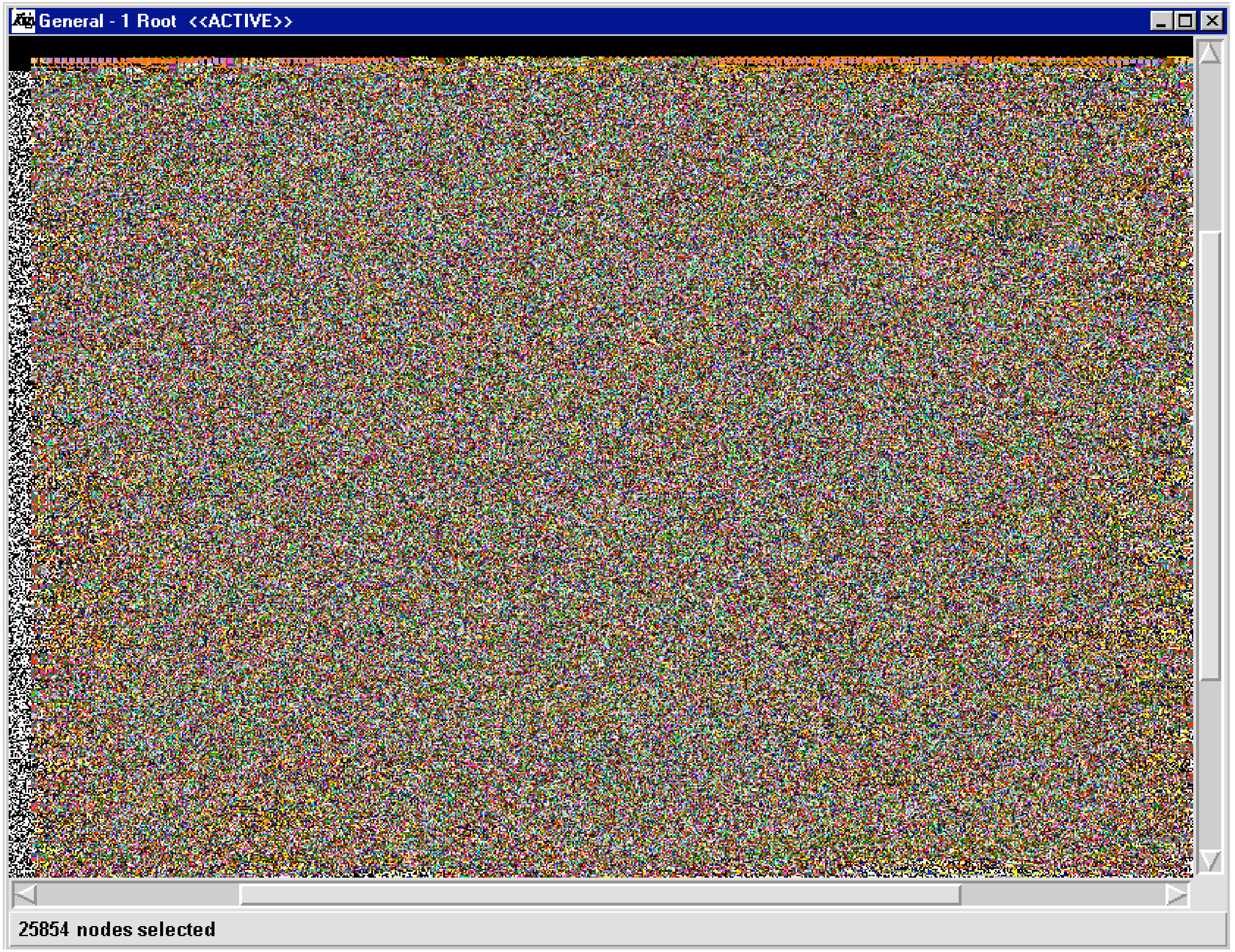
Why non-greenfield engineering?

Because existing software, often called legacy software, is valuable

- Often business-critical
- A huge amount of money has already been invested in it
- Has been tested to some extent and runs
- Does (mainly) what it should do

Would you replace such a system?

Why do we (often) start from a mess?



Recognizing the need to reengineer

- Obsolete or no documentation.
- Absence of thorough unit tests.
- Original developers or users have left.
- Inside knowledge of the system has disappeared.
- Limited understanding of the entire system.
- Too long to turn things over to production.
- Too much time to make simple changes.
- Need for constant bug fixes.
- Maintenance dependencies.
- Difficulties separating products.
- Long recompilation times.
- Duplicated code.
- Code smells.

Lehman’s Laws of Software Evolution

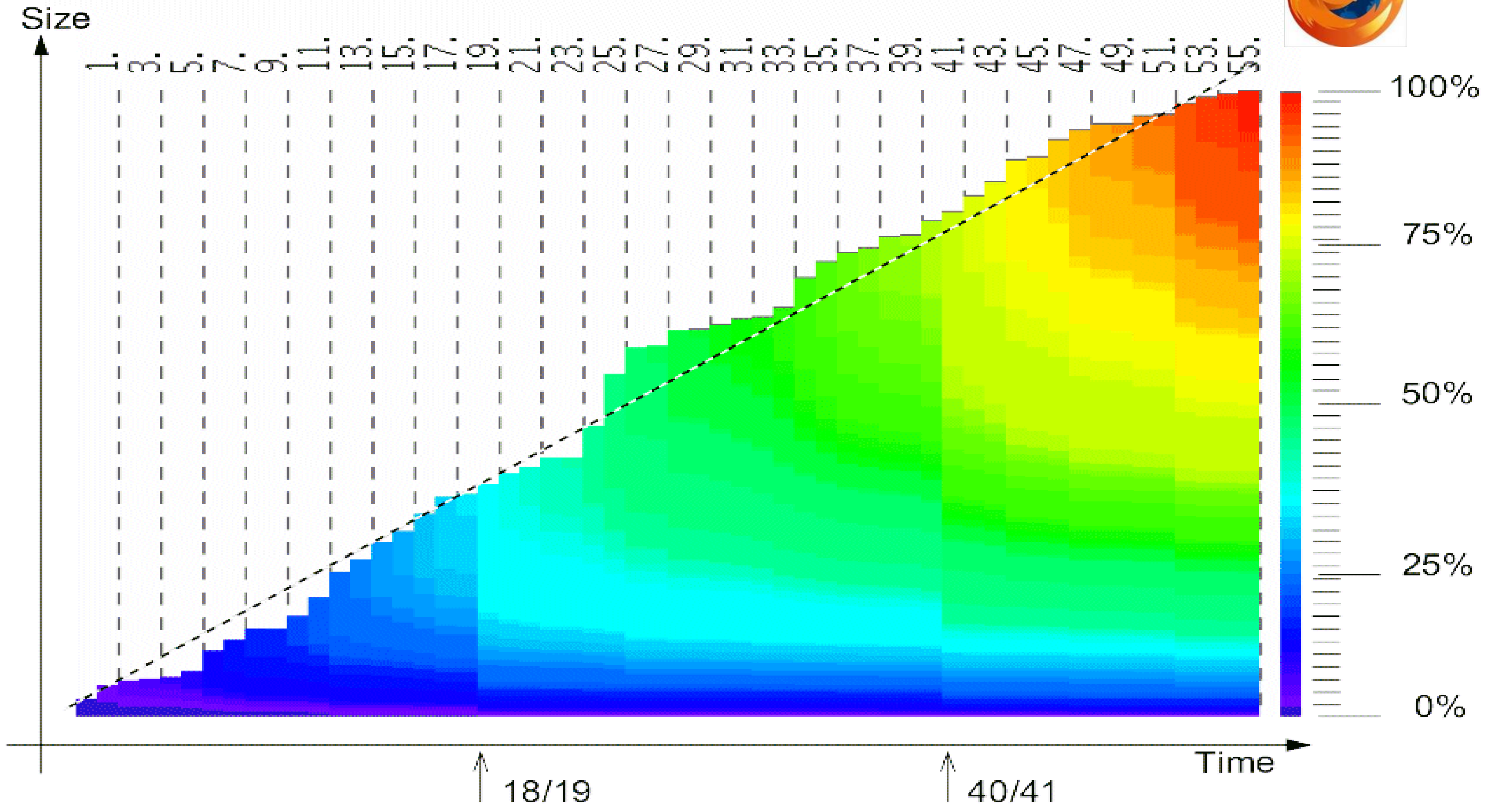
Continuing change

A program that is used in a real-world environment **must change**, as become progressively less useful in that environment.

Increasing complexity

As a program evolves, it **becomes more complex**, and extra resources are needed to preserve and simplify its structure.

Evolution of Mozilla source code



Lehman's Laws in practice

Existing software is often modified in an ad-hoc manner (quick fixes)

- Lack of time, resources, money, etc.

Initial good design is not maintained

- Spaghetti code, copy/paste programming, dependencies are introduced, no tests, etc.

Documentation is not updated (if there is one)

- Architecture and design documents

Original developers leave and with them their knowledge

Typical result of such practices

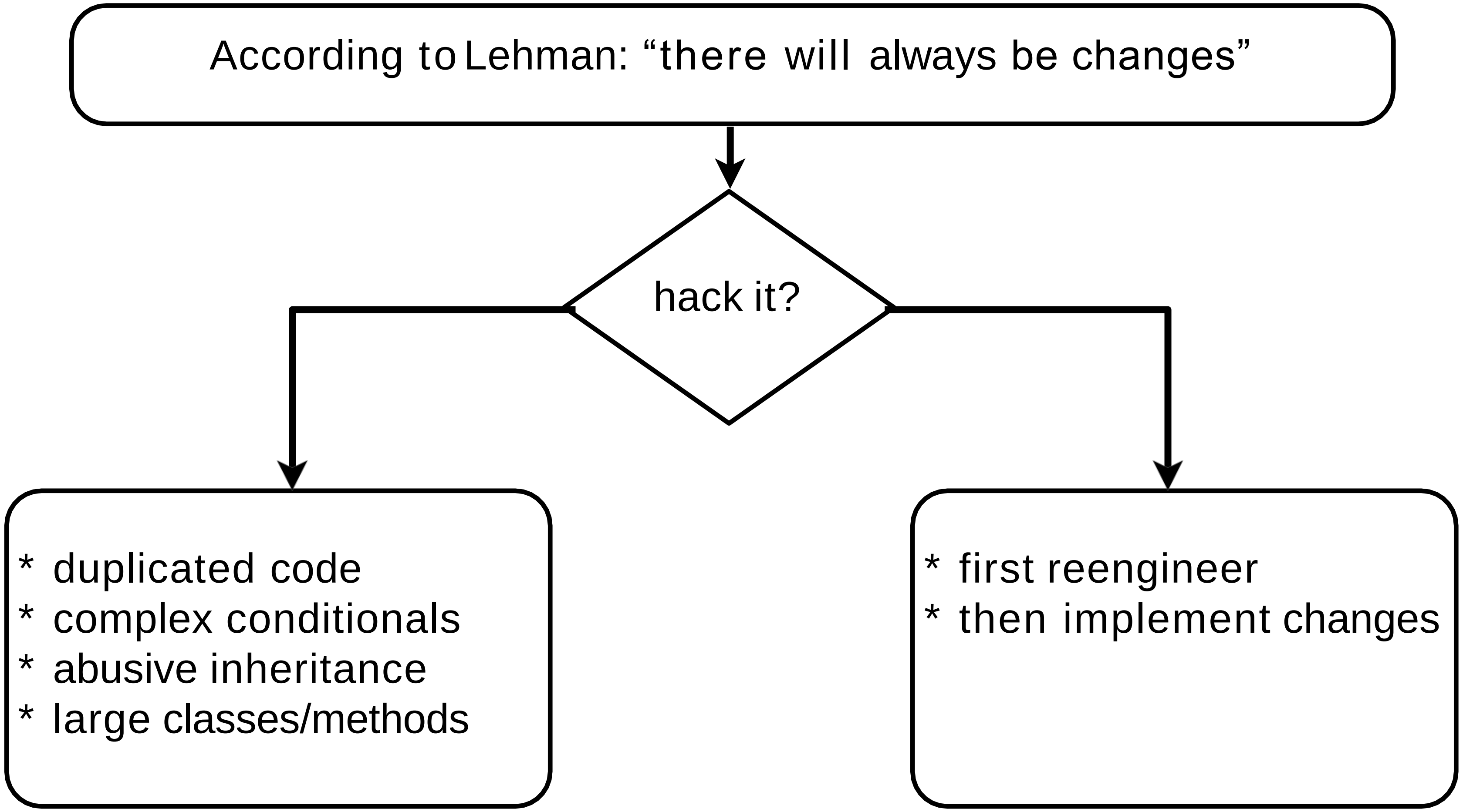
Implications of the results

Software maintenance costs continuously increase

Between 50% and 75% of global software development costs are spent on maintenance!

Up to 60% of a maintenance effort is spent on understanding the existing software

What is your decision?



Take a loan on your software
pay back via reengineering

Investment for the future
paid back during maintenance

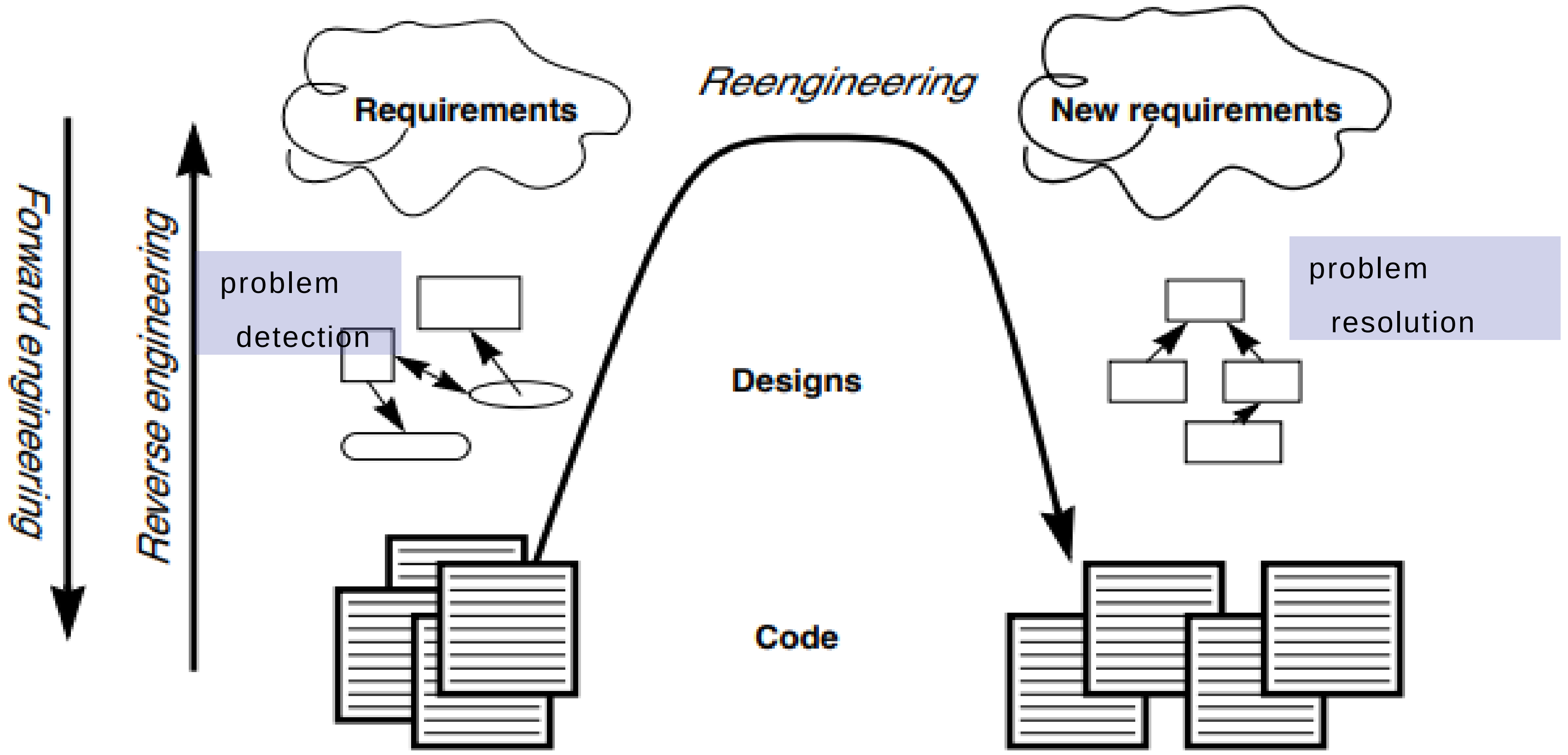
Let’s reengineer

Definition:

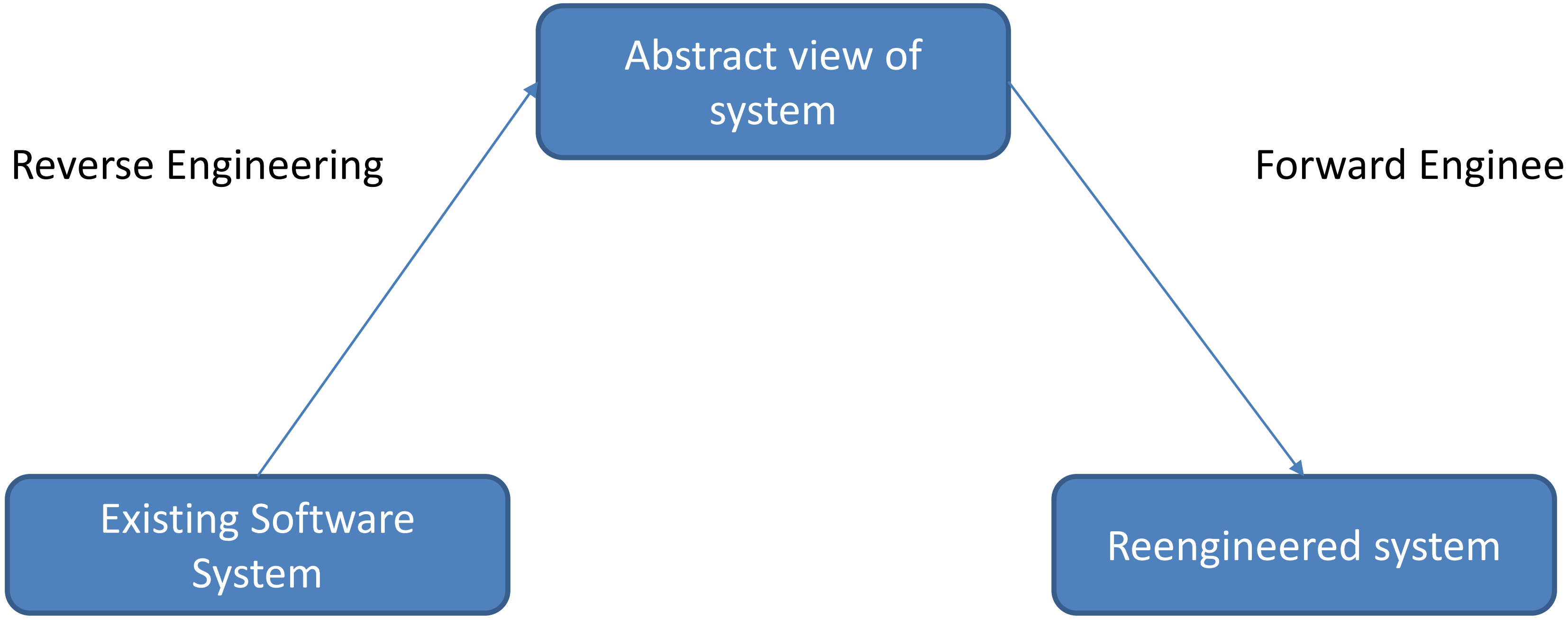
“Reengineering is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.”

[Demeyer, Ducasse, Nierstrasz]

Reengineering Life-Cycle



Reengineering Life-Cycle



Goals of reengineering

Testability

Understandability

Modifiability

Extensibility

Maintainability

…

Goals of reengineering

Unbundling

- Split a monolithic system into parts that can be separately marketed

Performance

- “First do it, then do it right, then do it fast”

Design extraction

- To improve maintainability, portability, etc.

Exploitation of New Technology

- I.e., new language features, standards, libraries, etc.

Reduce human dependencies

Reengineering Process Model

Reengineering is a rebuild activity

Consider the situation.

You’ve purchased a house in another state. You’ve never actually seen the property, but you acquired it at an amazingly low price, with the warning that it might have to be completely rebuilt. How would you proceed?

Reengineering is a rebuild activity

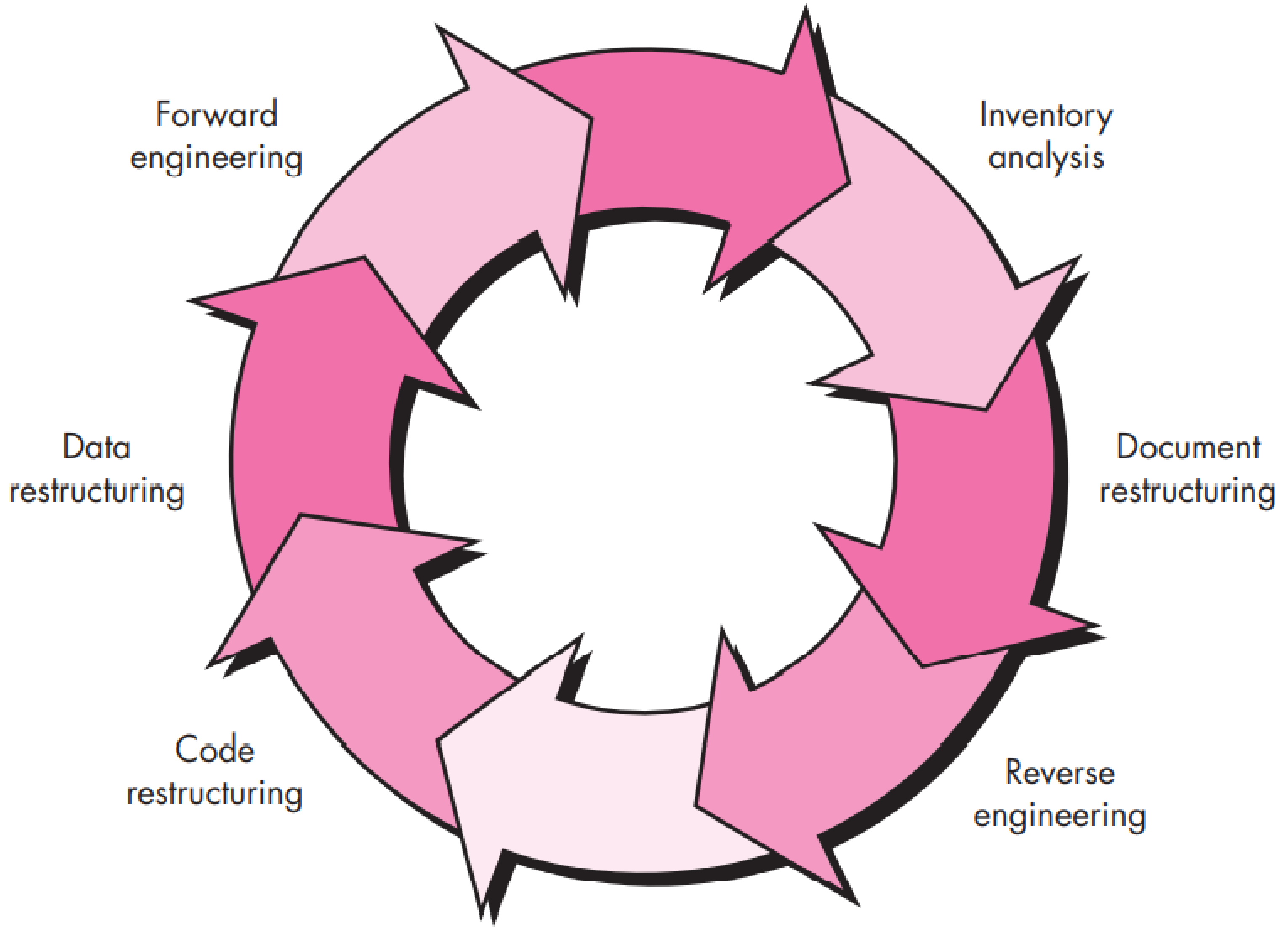
• Before you can start rebuilding, it would seem reasonable to inspect the house to determine whether it is in need of rebuilding.

- Before you tear down and rebuild the entire house, be sure that the structure is weak. If the house is structurally sound, it may be possible to “remodel” without rebuilding at much lower cost and in much less time.
- Before you start rebuilding be sure you understand how the original was built. Understand the wiring, the plumbing, and the structural internals. Even if you trash them all, the insight you’ll gain will serve you well when you start construction.

Reengineering is a rebuild activity

- If you begin to rebuild, use only the most modern, long-lasting materials. This may cost a bit more now, but it will help you to avoid expensive and time-consuming maintenance later.
- If you decide to rebuild, be disciplined about it. Use practices that will result in high quality—today and in the future.

Reengineering Process Model



Reengineering Activities

Inventory Analysis

Every software organization should have an inventory of all applications.

The inventory can be nothing more than a spreadsheet model containing information that provides a detailed description (e.g., size, age, business criticality) of every active application.

By sorting this information according to business criticality, longevity, current maintainability and supportability, and other locally important criteria, candidates for reengineering appear.

Resources can then be allocated to candidate applications for reengineering work.

It is important to note that the inventory should be revisited on a regular cycle.

The status of applications (e.g., business criticality) can change as a function of time, and as a result, priorities for reengineering will shift.

Document Restructuring

Creating documentation is far too time consuming.

Documentation must be updated, but your organization has limited resources:

Document when touched approach.

The system is business critical and must be fully redocumented.

Reverse Engineering

The term reverse engineering has its origins in the hardware world.

A company disassembles a competitive hardware product in an effort to understand its competitor’s design and manufacturing “secrets.”

These secrets could be easily understood if the competitor’s design and manufacturing specifications were obtained. But these documents are proprietary and unavailable to the company doing the reverse engineering.

In essence, successful reverse engineering derives one or more design and manufacturing specifications for a product by examining actual specimens of the product.

Reverse Engineering

Reverse engineering for software is quite similar.

In most cases, however, the program to be reverse engineered is not a competitor’s. Rather, it is the company’s own work (often done many years earlier). The “secrets” to be understood are obscure because no specification was ever developed.

Therefore, reverse engineering for software is the process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code.

Reverse engineering is a process of **design recovery**.

Code Restructuring

The most common type of reengineering is code restructuring.

Some legacy systems have a relatively solid program architecture, that individual modules were coded in a way that makes them difficult to understand, test, and maintain.

In such cases, the code within the suspected modules can be restructured.

To accomplish this activity, the source code is analyzed using a restructuring tool. Violations of structured programming constructs are noted and code is then restructured or even rewritten in a more modern programming language.

The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced.

Data Restructuring

A program with weak data architecture will be difficult to adapt and enhance.

Data restructuring is a **full-scale reengineering activity** that begins with a reverse engineering activity.

Current data architecture is dissected, and necessary data models are defined.

Data objects and attributes are identified, and existing data structures are reviewed for quality.

Forward Engineering

Traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.

Forward engineering not only recovers design information from existing software but uses this information to alter or reconstitute the existing system in an effort to improve its overall quality.

In most cases, reengineered software reimplements the function of the existing system and also adds new functions and/or improves overall performance.

The End