# CS266 Software Reverse Engineering (SRE)

## Introduction to Software Reverse Engineering

What's changed?

Teodoro (Ted) Cipresso
Senior Software Engineer, IBM
SRE Guest Lecture

The information in this presentation is partially taken from the thesis "Software reverse engineering education" available at http://scholarworks.sjsu.edu/etd_theses/3734/.

# Introduction to Software Reverse Engineering

- From very early on in life we engage in constant investigation of existing things to understand how and even why they work.
- Software Reverse Engineering (SRE) calls upon this investigative nature when one needs to learn how and why, often in the absence of adequate documentation, an existing piece of software—helpful or malicious—works.
- More formally, SRE can be described as the practice of analyzing a software system to create abstractions that identify the individual components and their dependencies, and, if possible, the overall system architecture.
- Once the components and design of an existing system have been recovered, it becomes possible to repair and even enhance/replace them.

# Introduction to Software Reverse Engineering
## (cont'd)

- In the early nineties, the Y2K problem spurred the need for the development of tools that could read large amounts of source or binary code for the 2-digit year vulnerability.
- in the mid to late nineties, the adoption of the Internet by businesses brought about the need to understand in-house legacy systems so that the information held within them could be made available on the Web.

What's changed?

- Today, expertise in legacy programming languages and systems is becoming scarce, prompting the creation and use of coding assistants that leverage generative AI for modernization efforts.
- For example, IBM's watsonx Code Assistant for Z (WCA4Z) facilitates the conversion of COBOL code into equivalent Java, including subsystem-specific elements like CICS, IMS, and Db2.

# Introduction to Software Reverse Engineering
## (cont'd)

What's changed?

- Today's technology often becomes tomorrow's legacy system, emphasizing the importance of good documentation for all software. However, documentation alone cannot fully eliminate the need for SRE.
- The vision is to incrementally incorporate Software Reverse Engineering (SRE) into normal development, or "forward engineering," to ensure critical system details—such as architecture, design constraints, and trade-offs—are well-documented and not confined to a developer's memory [1].
- Today, code explanation tools powered by generative AI are being used to document software either during development or retroactively.
- For example, IBM's watsonx Code Assistant can explain and document both legacy and modern code written in various programming languages.

# Reverse Engineering in Software Development

- While a great deal of software that has been written is no longer in use, a considerable amount has survived for decades and continues to run the global economy.
- The reality of the situation is that 70% of the source code in the entire world is written in COBOL. Compounding the situation is the fact that a great deal of legacy code is poorly designed and documented.
- COBOL programs are in use globally in governmental and military agencies, in commercial enterprises, and on operating systems such as IBM's z/OS®, Microsoft's Windows®, and the POSIX families (Unix/Linux etc.) [6].

What's changed?

- Enterprises are rapidly identifying low- to medium-risk applications that can be translated into modern languages or platforms with the help of generative AI.

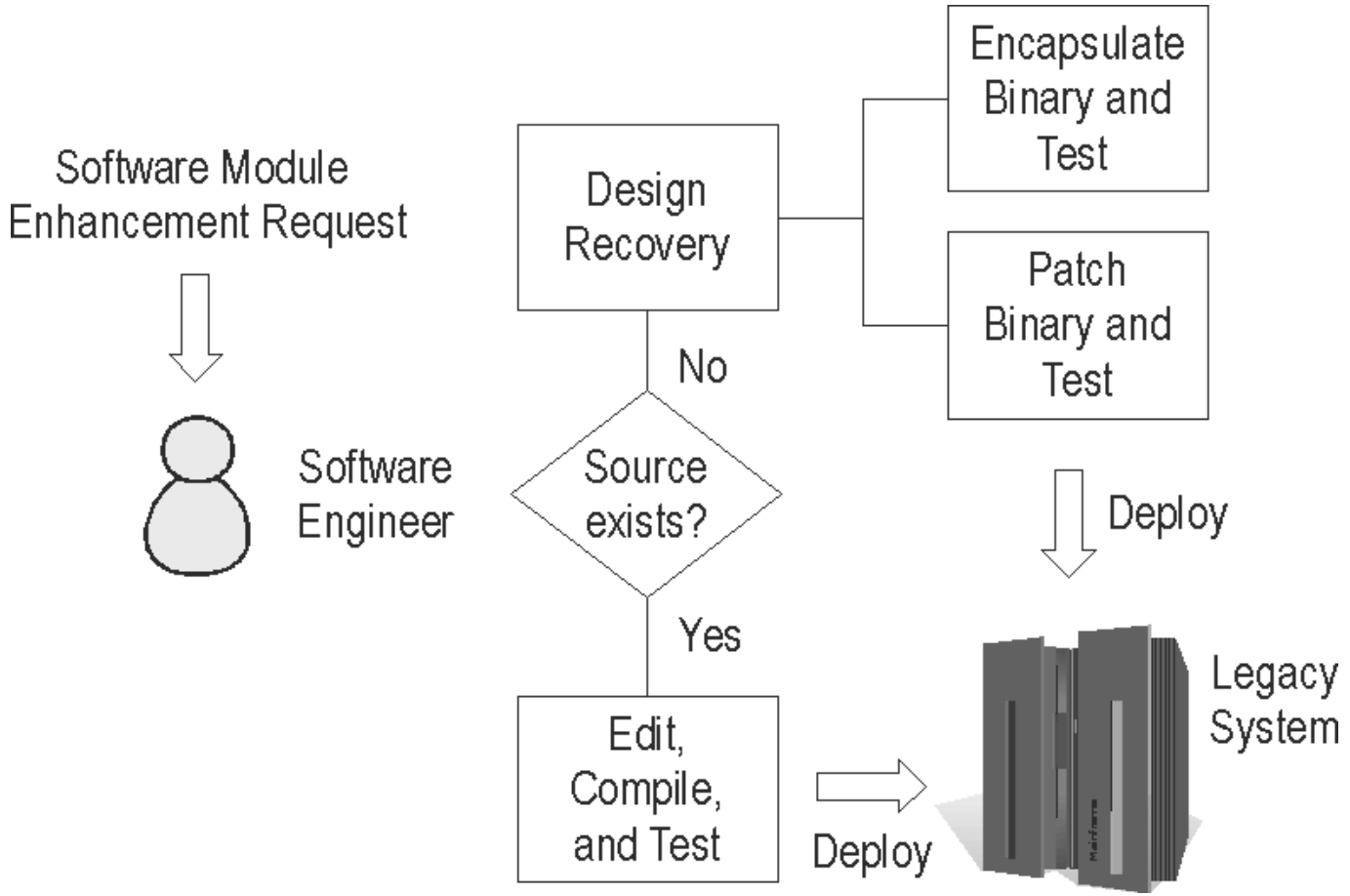
# Reverse Engineering in Software Development (cont'd)

- In 1997, the Gartner Group reported that 80% of the world's business ran on COBOL with over 200 billion lines of code in existence and with an estimated 5 billion lines of new code annually [6]. More recently…
- [http://simplicity.laserfiche.com/content/looking-job-hows-your-cobol]
  - This article from Aug 04, all suggests millennials learn COBOL ☺
  - COBOL supports 90 percent of Fortune 500 business systems every day.
  - 70 percent of all critical business logic and data is written in COBOL.
  - COBOL powers 85 percent of all daily business transactions processed.
  - 1.5 million new lines of COBOL code are written every day.
  - Do we have source code for all these applications?

# Reverse Engineering in Software Development (cont'd)

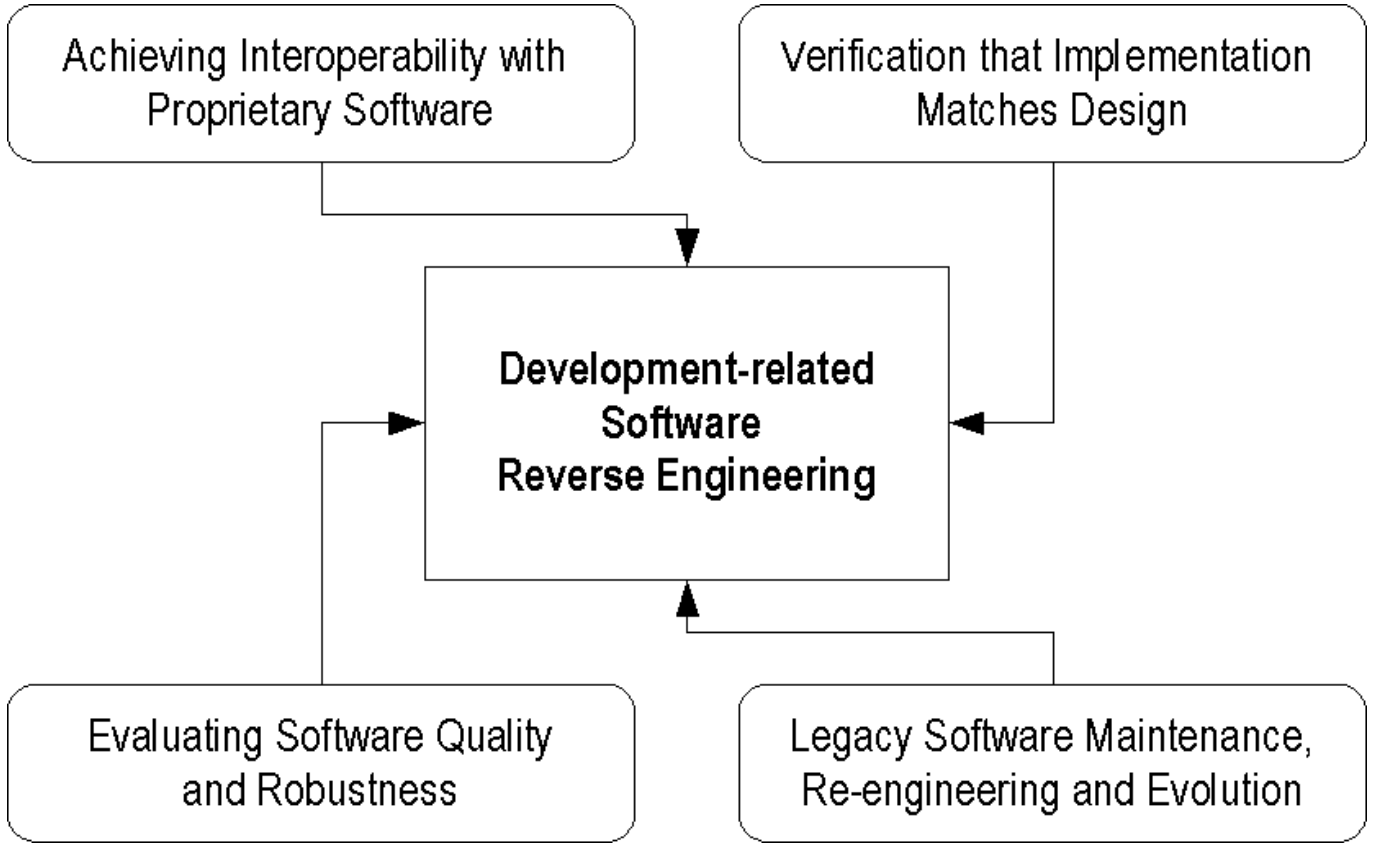
- Whenever computer scientists or software engineers are engaged with evolving an existing system, fifty to ninety percent of the work effort is spent on program understanding [3]…
  - "Practice with reverse engineering techniques improves ability to understand a given system quickly and efficiently."
- Even though several tools already exist to aid software engineers with the program understanding process, the tools focus on transferring information about a software system's design into the mind of the developer [1].
  - [4] states "commercial reverse engineering tools produce various kinds of output, but software engineers usually don't how to interpret and use these pictures and reports."

# Reverse Engineering in Software Development (cont'd)




**Software development process in a typical enterprise software system.**

# Reverse Engineering in Software Development (cont'd)




**Development-related software reverse engineering scenarios.**

# Reverse Engineering in Software Development (cont'd)

- Achieving Interoperability with Proprietary Software:
  - Develop applications or device drivers that interoperate (use) proprietary libraries in operating systems or applications.
- Verification that Implementation Matches Design:
  - Verify that code produced during the forward development process matches the envisioned design by reversing the code back into an abstract design.
- Evaluating Software Quality and Robustness:
  - Ensure the quality of software before purchasing it by performing heuristic analysis of the binaries to check for certain instruction sequences that appear in poor quality code.

# Reverse Engineering in Software Development (cont'd)

- Legacy Software Maintenance, Re-engineering, and Evolution:
  - Recover the design of legacy software modules when source is not available to make possible the maintenance, evolution, and reuse of the modules.

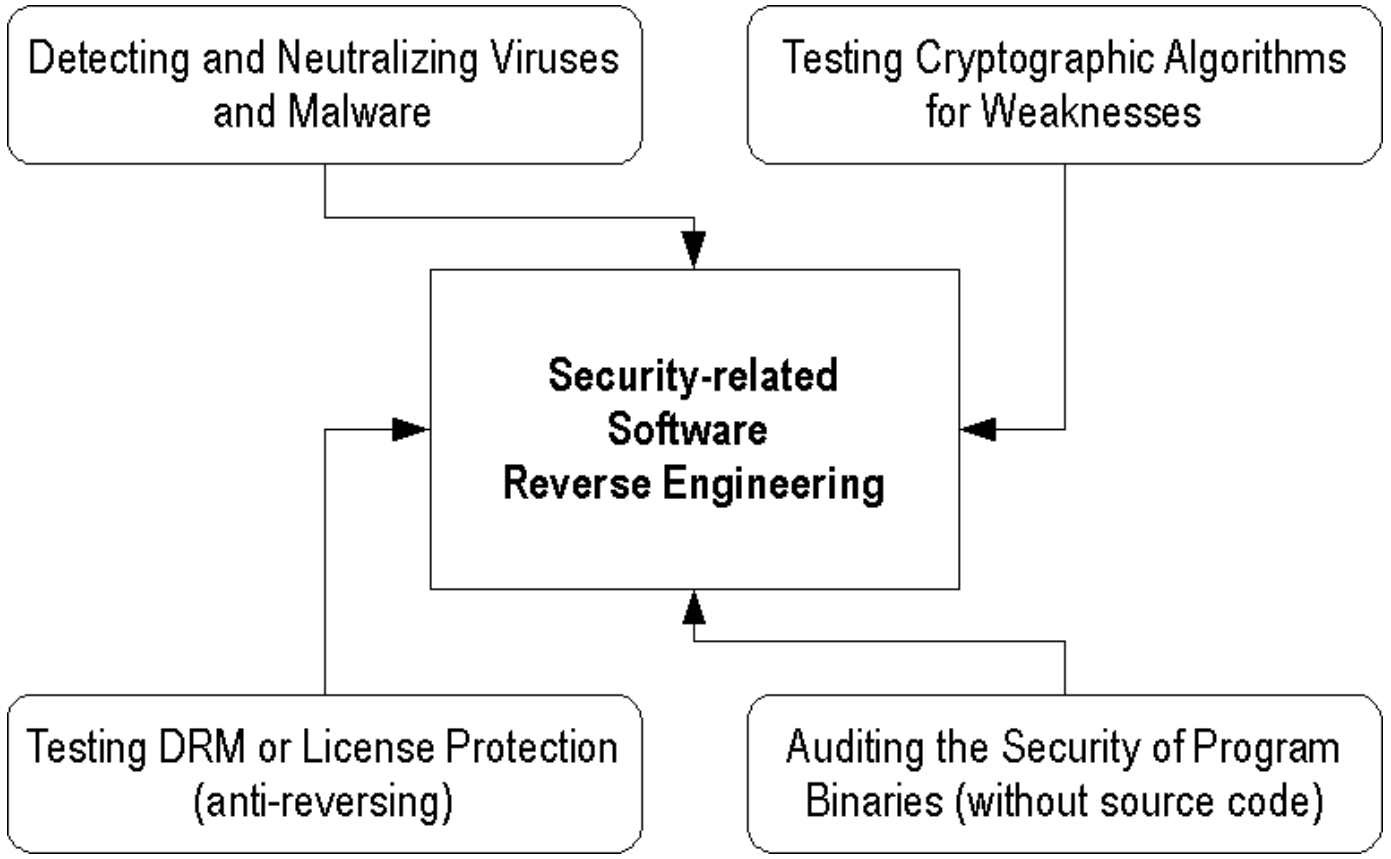
# Reverse Engineering in Software Development (cont'd)

- From the perspective of a software company, it is highly desirable that its products are difficult to pirate and reverse engineer.
  - Making software difficult to reverse engineer seems to conflict with the idea of being able to recover the software's design later for maintenance and evolution.
  - Manufacturers usually don't apply anti-reverse engineering transformations to software binaries until it is packaged for shipment to customers.
  - invest time in making software difficult to reverse engineer if there are algorithms that make the product stand out from the competition.
- Making software difficult to pirate or reverse engineer is often a moving target and requires special skills and understanding on the part of the developer.

# Reverse Engineering in Software Security (cont'd)

- [3] "to defeat a crook you have to think like one."
  - By reverse engineering viruses or other malicious software, programmers can learn their inner workings and witness first-hand how vulnerabilities find their way into computer programs.
- Interpreted languages like Java, JavaScript, Python…, which do not require programmers to manage low-level system details, have become ubiquitous.
  - In favor of productivity, programmers have increasingly lost touch with what happens in a system during execution of programs.

# Reverse Engineering in Software Security (cont'd)




**Security-related software reverse engineering scenarios.**

# Reverse Engineering in Software Security (cont'd)

- Detecting and Neutralizing Viruses and Malware:
  - Detect, analyze, or neutralize (clean) malware, viruses, spyware, and adware.
- Testing Cryptographic Algorithms for Weaknesses:
  - Test the level of data security provided by a given cryptographic algorithm by analyzing it for weaknesses.
- Testing DRM or License Protection (anti-reversing):
  - Protect software and media digital-rights through application and testing of anti-reversing techniques.

# Reverse Engineering in Software Security (cont'd)

- Auditing the Security of Program Binaries:
  - Audit a program for security vulnerabilities without access to the source code by scanning instruction sequences for potential exploits.

# End