Forense para Concursos de TI

Gustavo Pinto Vilar

Gustavo Vilar – Mini CV

- PCF / DPF – Perito Criminal Federal
- Pós-Graduado em Docência do Ensino Superior – UFRJ
- Graduado em Ciência da Computação e Processamento de Dados – ASPER/PB
- Aprovações: PRF 2002, PPF-PF 2004, PCF-PF 2004, MPU 2010, ABIN 2010, PCF-PF 2013

Gustavo Vilar

- Contatos:

http://www.itnerante.com.br/profile/GustavoPintoVilar

http://www.provasdeti.com.br/index.php/por-professor/gustavo-vilar.html

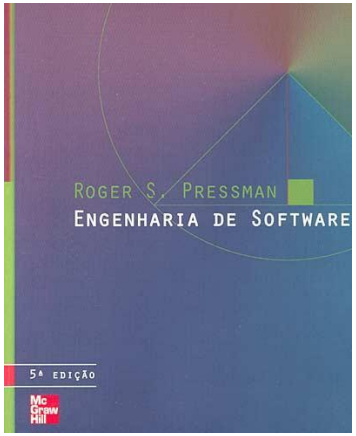
gustavopintovilar@gmail.com

p3r1t0f3d3r4l@yahoo.com.br

Escopo

- Abordar os assuntos mais recorrentes e com fortes tendências para concursos atuais
- Familiarizar o concursando com os tipos de questões mais frequentes.
- Abordar as metodologias de resolução de questões das principais bancas

Bibliografia

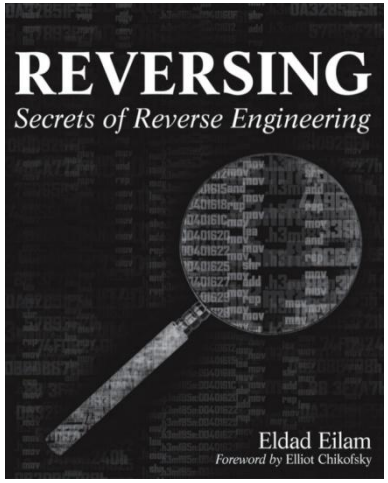


A Survey of Reverse Engineering Tools for the 32-Bit Microsoft Windows Environment

RAYMOND J. CANZANESE, JR., MATTHEW OYER, SPIROS MANCORIDIS, and MOSHE KAM
College of Engineering
Drexel University, Philadelphia, PA, USA

The Art of Unpacking

Mark Vincent Yason
Malcode Analyst, X-Force Research & Development
IBM Internet Security Systems
email: myason@us.ibm.com



cespeUnB

FUNDAÇÃO CESGRANRIO

Google

Fundação Carlos Chagas

ITnerante

PROVAS DE TI
TUDO PARA VOCÊ PASSAR

Forense para Concursos de TI – Carga Horária

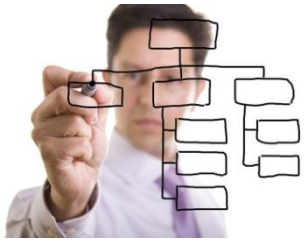
- **12 vídeo aulas (04h15m34s / 00h21m18s)**
  - Embasamento teórico
  - Conceitos afetos à ER
  - Realidade do software, aplicando ER, etapas da ER, propósitos e ferramentas
  - Reversão de códigos windows, etapas e passos na reversão, impedindo a ER
  - Análise de malware (introdução)
  - Análise estática, dinâmica e post-mortem
  - Red pointing, anti análise, contorno da anti análise
  - Compressão e descompressão de executáveis
  - Ferramentas CASE, Técnicas anti ER, refatoração, patching
  - Duas baterias de questões de aprendizagem

Forense para Concursos de TI

Embasamento Teórico

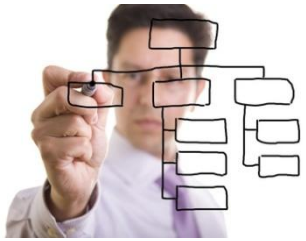
Engenharia de Software – Conceito

- Abordagem sistemática e disciplinada para o desenvolvimento de *software*
- Disciplina do conhecimento humano que tem por objetivo definir e exercitar processos (humanos atuando como máquinas), métodos (planos de processos), ferramentas e ambientes (máquinas apoiando processos e métodos) para construção de software que satisfaça necessidades de clientes e usuários dentro de prazos e custos previsíveis.

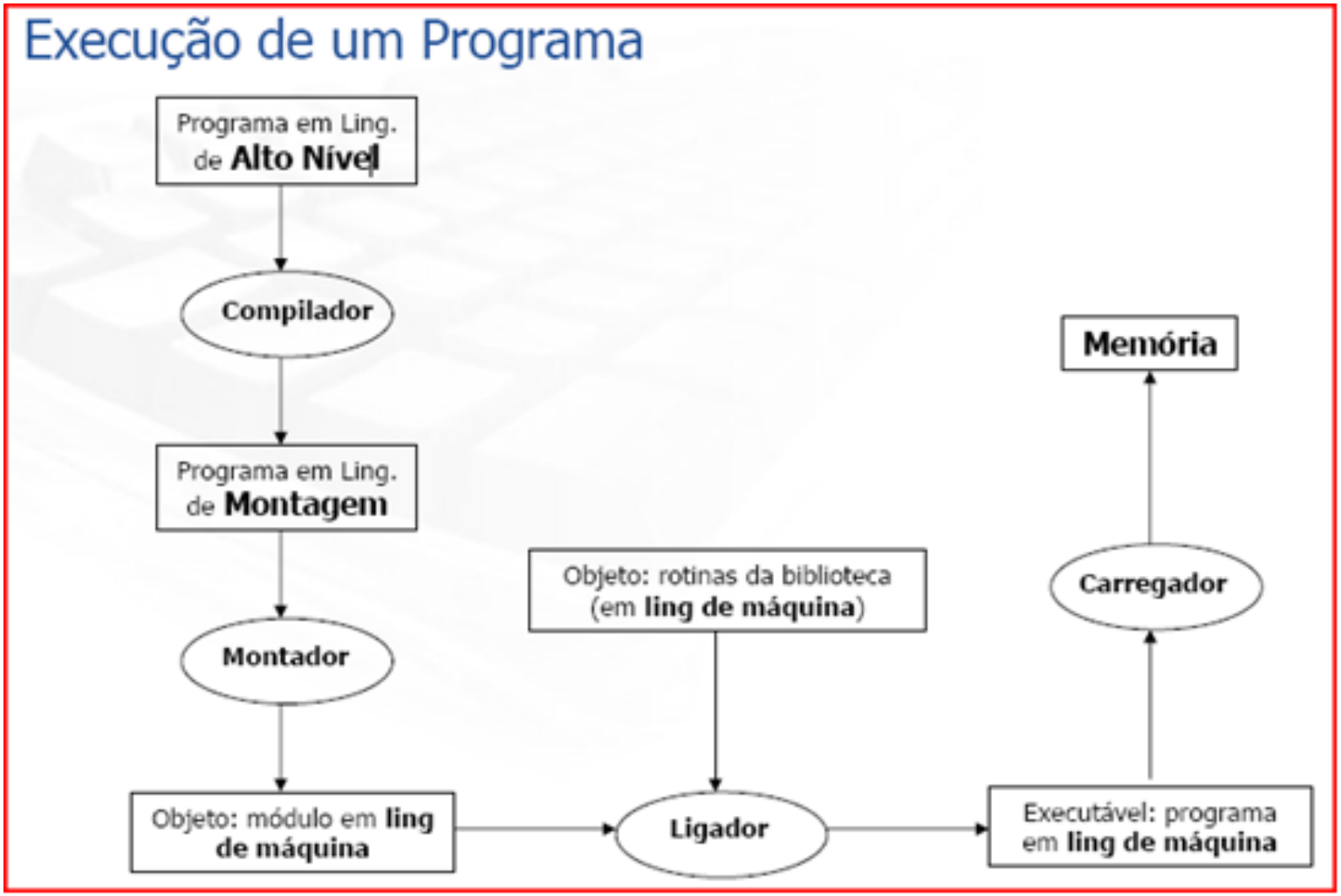


Engenharia de Software – Etapas da Engenharia Progressiva

- Especificação
- Modelagem
- Codificação
- Geração do Executável
- Manutenção



Panorama da Geração do Executável



Elementos da Engenharia Progressiva

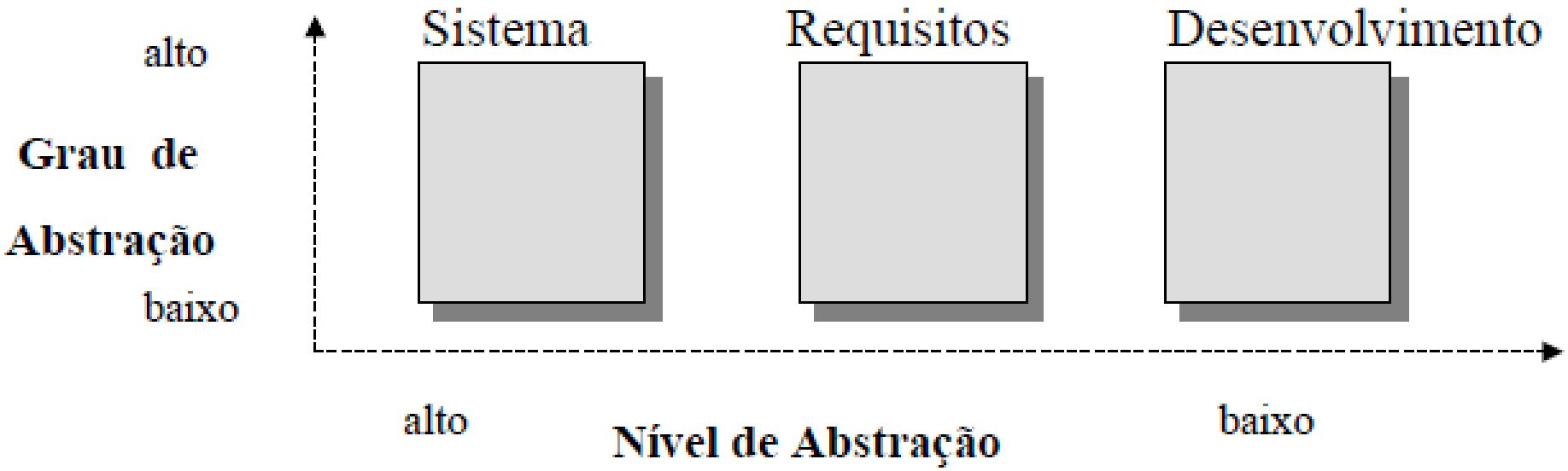
- Nível x Grau de abstração
- Completitude do processo
- Interatividade
- Direcionalidade

Abstração

- Habilidade de se ignorar os aspectos de assuntos não relevantes para o propósito em questão
- Conforme o nível de abstração aumenta, mais compreensíveis se tornam as informações

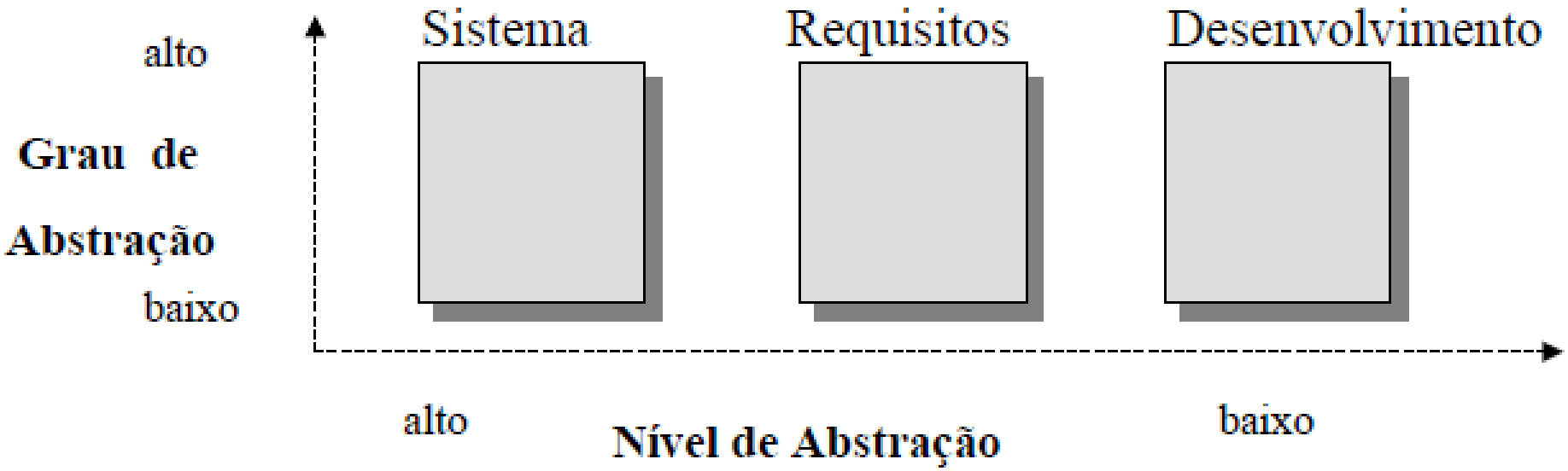
Abstração

- Estágios iniciais - alto nível de abstração
- Estágios finais - baixo nível de abstração
- Nível de abstração <> Grau de abstração
  - Nível = fases diferentes
  - Grau = mesma fase



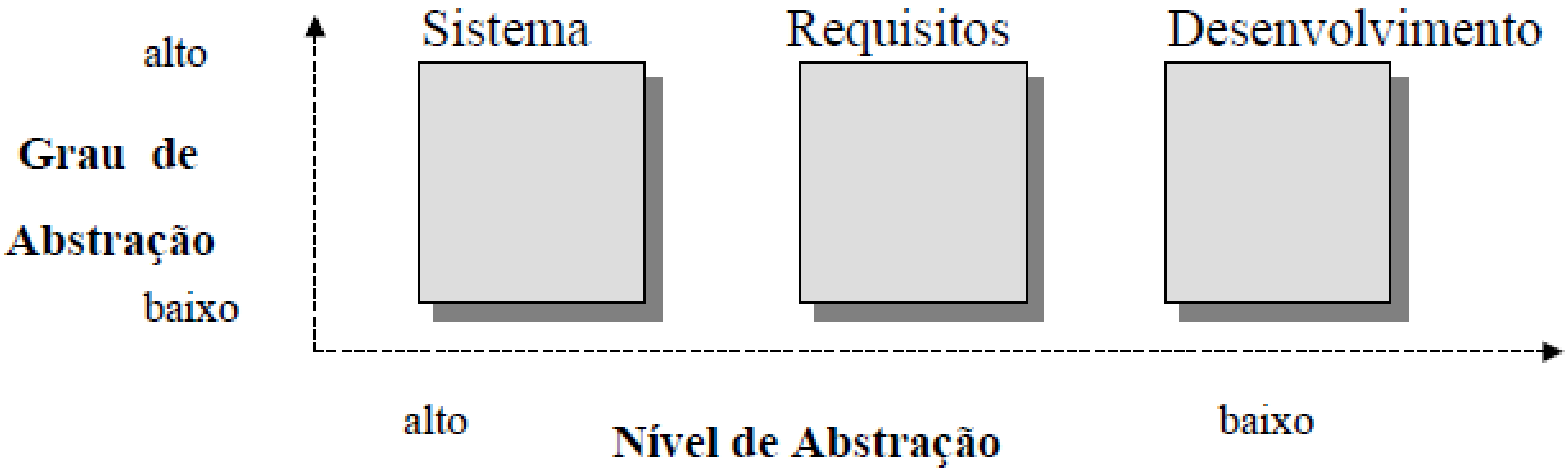
COMPLETITUDE DO PROCESSO

- Refere-se à quantidade de detalhe que é fornecido **em cada nível** de abstração



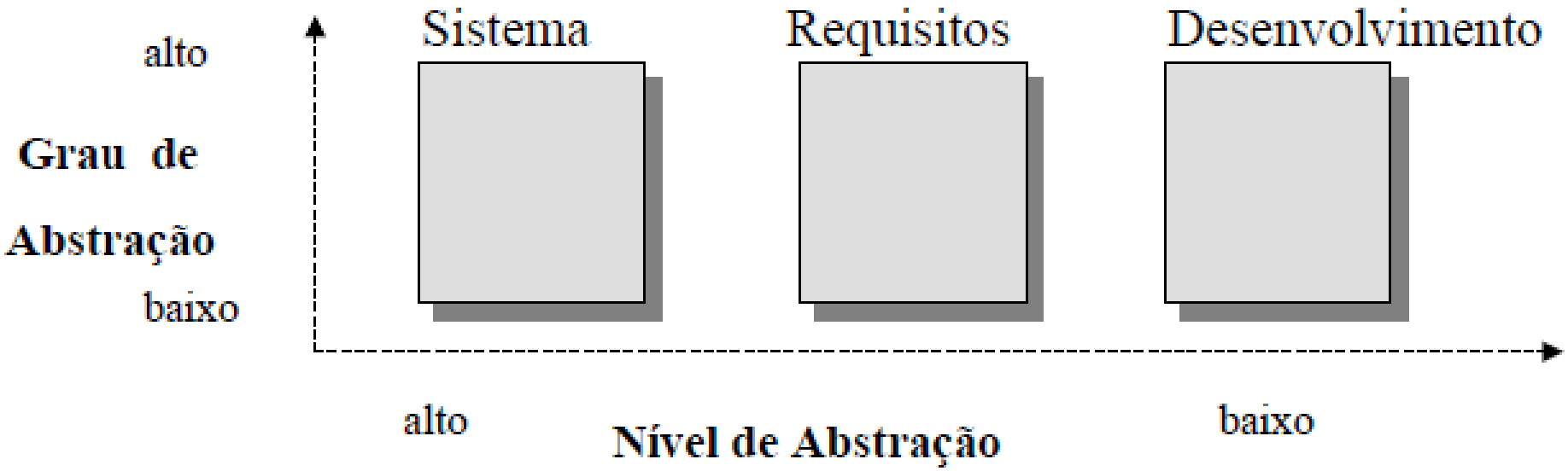
INTERATIVIDADE

- Refere-se ao grau de participação do ser humano no processo de engenharia
- Conforme o nível de abstração aumenta, a interatividade deve aumentar ou a completitude será prejudicada



DIRECIONALIDADE

- Se a direcionalidade tem sentido único, toda informação extraída a partir do código fonte é usada durante as atividades de manutenção
- Se a direcionalidade tem sentido duplo, a informação é usada para "alimentar" uma ferramenta de reengenharia



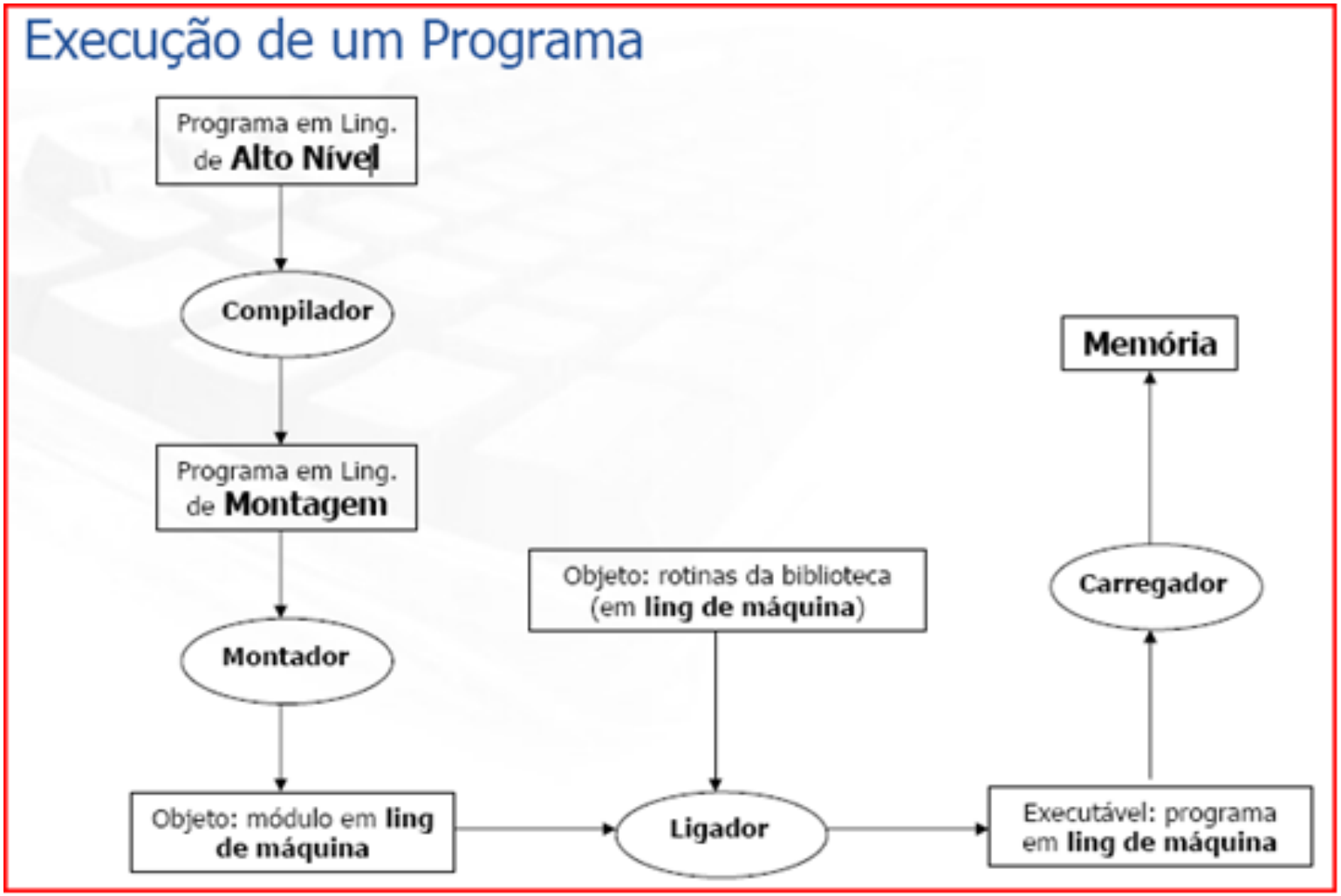
Compreensão da Engenharia Reversa

- Engenharia Progressiva
  - Processo tradicional de engenharia de software
  - Partem de um alto nível de abstração, para um baixo nível de abstração
  - Sinônimo - Engenharia avante

Compreensão da Engenharia Reversa

- É o fluxo inverso da engenharia progressiva
- O processo inverso a Engenharia Progressiva

Engenharia Reversa



Compreensão da Engenharia Reversa

- Do nível mais baixo de abstração para o nível mais alto
  - Caracterizado pelas atividades retroativas do ciclo de vida,
  - Pode ser de programas ou de dados

Compreensão da Engenharia Reversa

- Termo surgiu na análise de hardware
  - Entender um motor
  - Copiar uma calça jeans
- No mundo do software
  - Drivers
  - SO Linux
  - Auditoria de sistemas

Compreensão da Engenharia Reversa

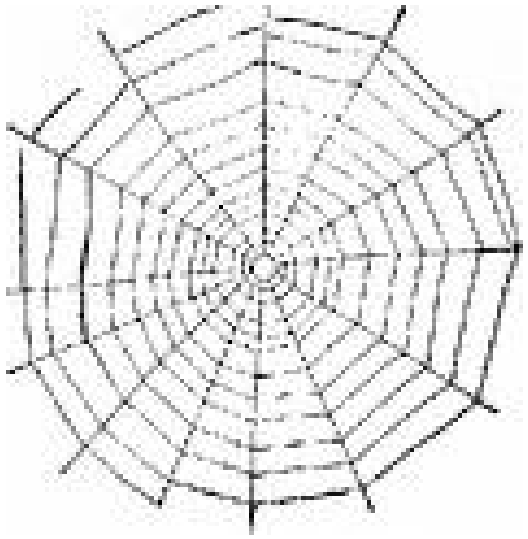
- Benefício econômico
  - Aumento do entendimento de um sistema
  - Facilita novos desenvolvimentos
  - ER nem sempre viola a propriedade intelectual

Compreensão da Engenharia Reversa

- Motivações para ER
  - Hacking
  - Desbloqueios
- Dificuldades
  - Encontrar rotinas de validação
  - Inversão de testes lógicos
  - Não corromper mecanismos de teste de integridade

Sistema Legado

- Apresenta inconformidades em algum dos itens:
  - Tempo de vida
  - Utilidade
  - Tecnologias
  - Manutenção
  - Documentação



Transliteração

- Ato de traduzir o código de um programa de uma linguagem para outra, ou para uma versão mais nova da mesma linguagem

Reengenharia

- Ação de analisar e modificar um sistema para recriá-lo e reimplementá-lo com nova estrutura

Passos da Reengenharia

- Engenharia Reversa
- Estudo das possibilidades existentes
- Reengenharia

Reengenharia pode ser

- Sem mudança de funcionalidade
  - Mudança da linguagem de programação por exemplo

Reengenharia pode ser

- Mudança parcial de funcionalidade
  - Integração entre as partes
  - Uso de linguagens compatíveis
- Mudança total de funcionalidade

Artefatos produzidos na engenharia de software

- Fase de análise (o que o sistema deve fazer)
  - Documento de especificação
- Projeto (Como fazer)
  - Documento de arquitetura
- Implementação
  - Código

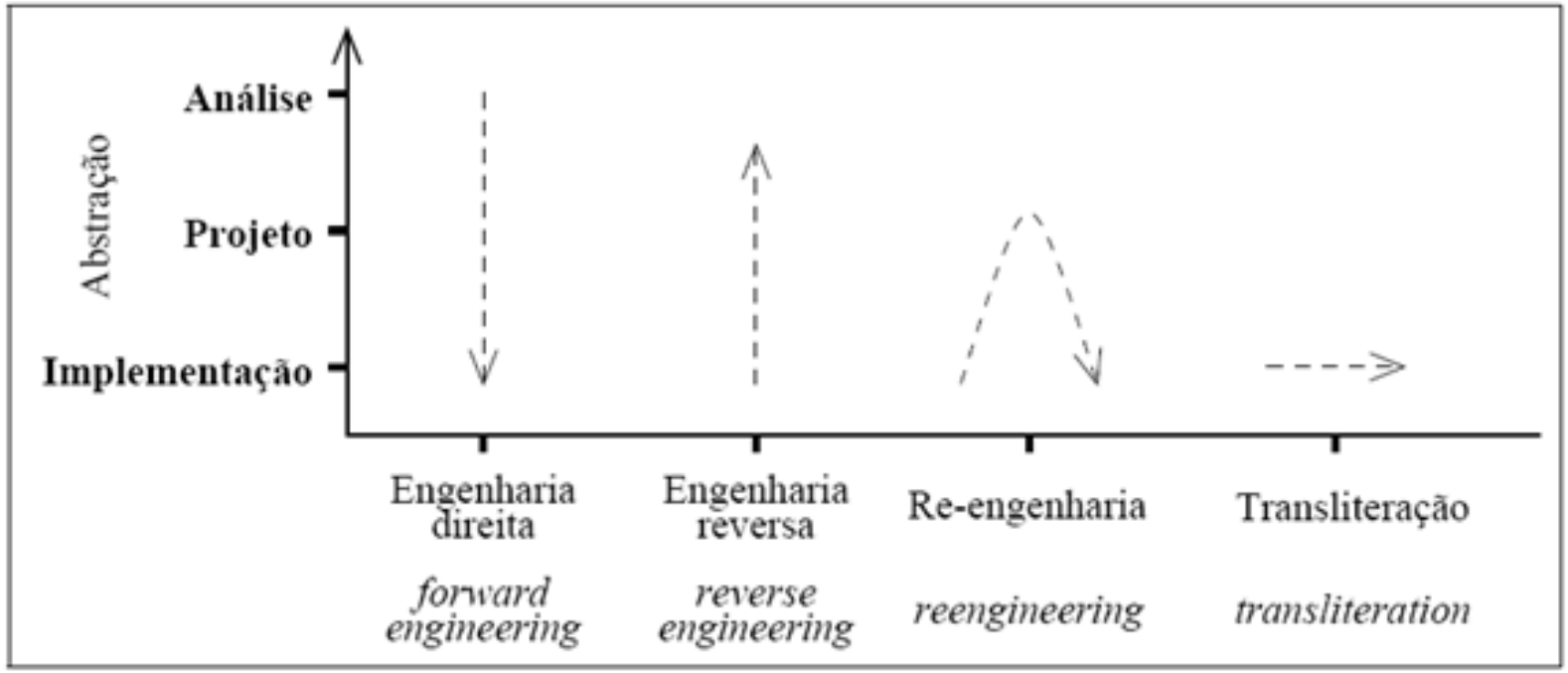


Figura 1.1: Representação gráfica de várias atividades na engenharia de software.

Realidade do Software

- Sistemas com documentação insuficiente / inexistente
- Código duplicado ou mal escrito
- Dificuldade de manutenção
- Erros gerando outros erros

Realidade do Software

- Fase de manutenção de software
  - sistema entra num estado contínuo de mudanças
  - Sistemas vão se tornando obsoletos, em função de novas tecnologias

Realidade do Software

- Tipos
  - Migrações para novas plataformas
  - Ajustes para mudanças de tecnologia de hardware ou S.O.
  - Extensões em sua funcionalidade

Realidade do Software

- Essas mudanças produzem
  - Estruturas mal projetadas
  - Documentação desatualizada
  - Lógica e codificação ruins

Realidade do Software

- Solução
  - Engenharia reversa
  - Reengenharia

Onde aplicar o que esperar da ER

- Engenharia reversa de interfaces
- Redocumentação
  - Intenção de recuperar a documentação que já existiu ou deveria ter existido
- Modularização
- Descoberta de anomalias no código
- Etc...
  - Todas as etapas podem ser divididas em 3 etapas

Etapas

1. Extração de fatos do sistema a analisar
2. Tratamento dos fatos
3. Visualização dos resultados

Extração dos fatos

- Coleta de informações sobre o sistema
- Análise Estática do Código – Parsing
- Análise Dinâmica do Código – Debugging
- Dados
  - Banco de dados do próprio sistema
- Documentação
  - É tudo o que não está usado pelo computador para fazer funcionar o sistema
  - Podem ser textos, diagramas, helps, etc

Extração dos fatos

- Outras fontes de informação
  - A linguagem de programação que foi utilizada
  - O sistema operacional
  - O tipo de processador, etc
- Tratamento dos fatos
  - A documentação do tratamento efetuado é fundamental
  - Uma vez que se tenha informações suficientemente claras e precisas, é possível alterar o código e até a estrutura de um sistema (ou programa) utilizando várias técnicas
  - Tentar abstrair informações de mais alto nível de abstração de fatos do sistema, ou seja, eliminação de detalhes
- Visualização dos resultados

Propósitos para Engenharia Reversa

- Manutenção (60 por cento do esforço)
  - Adaptativa
    - Adequar o software a um novo ambiente
  - Evolutiva
    - Adição de funcionalidades
  - Corretiva
    - sanar erros
  - Preventiva
    - Redução de esforços para futuras mudanças
- Reuso
  - Identificar software ou componentes reaproveitáveis
  - Reconhecimento, decomposição, classificação, seleção, adaptação e composição

Ferramentas utilizadas

- Ferramentas podem ser classificadas em
  - Estáticas
    - Análise dos códigos ou modelos
  - Dinâmicas
    - Análise do código em execução

Reversão de Código

- Em ambientes Windows:
  - Executável está disponível (exe, dll)
  - Linguagem não compreensível diretamente pelo ser humano
  - Temos capacidade de “pegar” no executável, logo revertê-lo

Passos na reversão de executáveis Windows

1. Identificação do código
   - Identificar o compilador que gerou o código
     - Uso do programa PEiD (Portable Executable iDentificator)
     - Reconhece vários compiladores
     - Reconhece os packers também
       - UPX é um packer muito usado
2. Efetuar a desmontagem/descompilação

Passos na reversão de executáveis Windows

3. Realizar as Análises
   - Análise estática básica
   - Análise dinâmica básica
   - Análise estática avançada
   - Análise dinâmica avançada

Análise Estática Básica

compreensão da estrutura do código

- Finalidade: Descobrir vínculos de APIs, DLLs e Strings
  - Muito lenta ou quase impossível, se realizada diretamente sobre o código binário

Análise Estática Avançada

Dead-Listing Analisys

- É a leitura de código propriamente dita
- OBS: A separação dos 2 tipos de análise estática é mais didática do que efetiva. Na prática, ocorrem em concomitância

Análise Dinâmica Básica

Análise comportamental

- Executar o código num **ambiente** seguro e controlado
- Ferramentas: System Monitor, Netstat, Wireshark, servidores reais para suprir as demandas

Análise Dinâmica Avançada

Execução dentro de um Debugger

- Análise de valores de variáveis
- Execução das funções
- Foco interno no executável

Passos na reversão de executáveis Windows

4. Identificação do ponto a ser resolvido
   - Reduzir o tamanho do problema
   - Identificação do código desejado
   - Uso de breakpoints
5. Correção do ponto chave

Como impedir a engenharia reversa?

- Colocar parte da funcionalidade do aplicativo em WebServices ou mesmo servidores de aplicativos (COM+/remoting). Como os usuários não têm acesso aos executáveis, não há como decompilá-los.
- Escrever parte do programa como código não gerenciado, quer como DLL, objeto COM ou mesmo em código não gerenciado dentro de programas C++ .NET.

Como impedir a engenharia reversa?

- Escrever parte do código em Assembly IL (intermediate language) usando recursos que dificultam a decompilação

Como impedir a engenharia reversa?

- Usar um “obfuscator” para alterar o código de forma a dificultar a compreensão do código após decompilação.

Efetuando a análise do malware

- Visa ao entendimento profundo do funcionamento de um software
  - Identificar tipos de técnicas de ofuscação usadas
  - Como atua no sistema operacional
  - Quais fluxos de execução levam ao comportamento principal planejado
  - Se há operações em rede
  - Download de outros arquivos
  - Captura de informações do usuário ou sistema, etc...

Efetuando a análise do malware

- As atividades realizadas no sistema vítima sempre podem ser mapeadas para ações específicas
  - Desabilitar mecanismos de proteção (firewall, antivírus)
  - Modificar binários e bibliotecas do sistema operacional
  - alterar chaves do registro
  - abrir portas para comunicação
  - etc...

Efetuando a análise do malware

- Todas as abordagens de análise possuem deficiências
  - Sucumbir a mecanismos de detecção presentes nos malware, que identificam a própria monitoração por terceiros
    - Terminam sua execução
    - Modificam o comportamento
      - Mais evidente na análise dinâmica

Efetuando a análise do malware

- Dependendo da técnica ou ferramenta que se utiliza para fazer cada análise, a velocidade pode variar, mas, em geral, análises estáticas simples são mais rápidas do que as dinâmicas.
- Se há a necessidade de engenharia reversa, se o software possui muitos fluxos de execução ou se está comprimido com um packer de difícil descompressão, a análise dinâmica tradicional é muito mais rápida e eficaz na provisão de resultados acerca do comportamento do exemplar analisado

Análise de Malware – Problemas dos Antivírus

- Um dos mecanismos de defesa contra malware mais populares ainda é o antivírus, que pode ser explicado basicamente como um programa que varre arquivos ou monitora ações pré-definidas em busca de indícios de atividades maliciosas
- O grande problema dos antivírus é o surgimento frequente e crescente de variantes de malware previamente identificados, cujas ações modificadas visam a evasão da detecção
- Essas variantes precisam ser tratadas muitas vezes individualmente (e manualmente) no caso da confecção de uma assinatura para um antivírus.
- Pode ser preciso modificar a heurística de detecção de toda uma classe para que esta seja capaz de identificar a variante

Análise de malware – Problemas dos Antivírus

- Operam de duas formas
  - Correspondência de padrões em bancos de dados de assinaturas
    - Um arquivo executável é dividido em pequenas porções (chunks) de código, as quais são comparadas com a base de assinaturas do antivírus
    - Se um ou mais chunks do arquivo analisado estão presentes na base de assinaturas, a identificação relacionada é atribuída ao referido arquivo

Análise de malware – Problemas dos Antivírus

- Heurísticas comportamentais
  - Arquivo sob análise é executado virtualmente em um emulador minimalista e os indícios de comportamento suspeito são avaliados a fim de se verificar se a atividade realizada pelo programa pode ser considerada normal ou um alerta deve ser emitido
  - Uma heurística bem construída pode resultar na substituição de dezenas de assinaturas

Análise de malware – Problemas dos Antivírus

- Ocorrência de falsos positivos
  - Identificar erroneamente uma aplicação inofensiva do usuário como sendo maliciosa e, consequentemente, removê-la, colocá-la em quarentena (bloqueio) ou restringir de algum modo a usabilidade de um sistema computacional

Análise de malware – Problemas dos Antivírus

- Muitos exemplares de malware possuem mecanismos próprios de defesa
  - Desabilitar as proteções existentes no sistema operacional alvo (firewall, antivírus, plugins de segurança)
  - Verificar se o exemplar está sob análise, mudando seu comportamento
  - Disfarçar-se de programas do sistema, inclusive de falsos antivírus
- Desenvolvedores de antivírus ainda não possuem padronização para a classificação dos malwares identificados por suas ferramentas

Análise estática de softwares maliciosos

- Objetivos
  - Obter informações sem necessidade de execução
  - Análise de strings, disassembling e engenharia reversa
- OBS: Ofuscação de malware prejudica esta análise

Análise estática de softwares maliciosos

- Técnicas
  - Localização de funções geradoras de hashes criptográficos
  - identificação das funções importadas e exportadas
  - identificação de código ofuscado
  - obtenção de cadeias de caracteres
    - mensagens de erro
    - URLs e endereços de correio eletrônico

Análise estática de softwares maliciosos

- Abordagens
  - Verificação de padrões no arquivo binário
    - Geração de sequência de bytes, chamadas de assinaturas, que identificam um trecho de código frequentemente encontrado em programas maliciosos e verifica-se se o programa possui tal sequência
  - Análise do código assembly gerado a partir do código de máquina do malware
    - Técnicas de análise mais profundas que buscam padrões de comportamento malicioso

Análise estática de softwares maliciosos

- Tenta-se derivar o comportamento do malware extraindo características de seu código sem executá-lo, através de análise de strings, disassembling e engenharia reversa, por exemplo
- Durante a análise estática não se sabe como o sistema vai reagir em resposta às operações do programa

Análise estática de softwares maliciosos

- Pode ser utilizada para obter informações gerais sobre o programa e para identificar a existência de código malicioso
- A maior dificuldade encontrada pela análise estática é o uso dos packers
  - Para combater a evolução destes, foram desenvolvidos diversos mecanismos que visam a obter o código não ofuscado do malware, permitindo que a análise estática seja efetuada

Análise estática de softwares maliciosos

- A análise estática tem como objetivo coletar informações presentes no malware sem ter que executá-lo
- Após o início da análise estática, a primeira informação buscada é uma chave que identifique univocamente o malware
  - Função de hash
  - Buscar cadeias de caracteres relevantes, com pelo menos 3 bytes, no código
  - Algumas identificações do malware podem ser coletadas com o uso de programas antivírus, identificadores de packer e identificadores de bibliotecas

Análise estática de softwares maliciosos

- As bibliotecas encontradas na análise estática não são necessariamente as mesmas utilizadas durante a execução do malware, assim é preciso checar quais foram carregadas em tempo de execução

Técnica estrutural da caixa branca

- Diferentemente dos testes funcionais , o teste estrutural enfoca a implementação e a estrutura da função
- É uma técnica de teste que usa a PERSPECTIVA INTERNA do sistema para modelar os casos de teste
  - Se software = código fonte
  - Se Hardware = teste em todos os nós

Técnica estrutural da caixa branca

- Geralmente usado durante a fase de codificação.
- A intenção é encontrar dados de teste que terão cobertura de todas estruturas presentes na representação formal
- Visa a conhecer o funcionamento interno de um produto e verificar que a sua operação tem um desempenho de acordo com as especificações

Técnica estrutural da caixa branca

- Verifica o comportamento INTERNO do SW, trabalhando diretamente sobre o código-fonte, avaliando as condições, os fluxos de dados, os ciclos, os caminhos lógicos e os caminhos nunca executados
- O testador tem acesso ao código fonte da aplicação e pode construir códigos para efetuar a ligação de bibliotecas e componentes
- Este tipo de teste é desenvolvido analisando o código fonte e elaborando casos de teste que cubram todas as possibilidades do componente de software

Análise dinâmica de softwares maliciosos

- Comportamento do malware é monitorado durante sua execução
- Ofuscação é ineficiente
- Malware é monitorado durante sua execução, por meio de emuladores, debuggers, ferramentas para monitoração de processos, registros e arquivos e tracers de chamadas de sistema

Análise dinâmica de softwares maliciosos

- Como opção aos emuladores limitados embutidos nos antivírus com o objetivo de realizar a identificação por heurísticas, os sistemas de análise dinâmica de malware foram sendo aprimorados
- Tais sistemas lançam mão de uma variedade de técnicas para monitorar a execução de um malware de maneira controlada, utilizando desde a instrumentação de emuladores complexos até a interceptação de chamadas ao kernel do sistema operacional monitorado

Análise dinâmica de softwares maliciosos

- É comum as empresas fabricantes de antivírus possuírem seus próprios sistemas de análise dinâmica de malware, também chamados de sandboxes
- Uma sandbox é um ambiente restrito e controlado que permite a execução de um código malicioso de forma a causar danos mínimos à sistemas externos por meio da combinação de filtragem e bloqueio de tráfego de rede e da execução temporizada do malware.

Análise dinâmica de softwares maliciosos em sandboxes

- O malware é executado por quatro ou cinco minutos e, durante este tempo, são monitoradas as ações pertinentes tanto ao malware quanto aos processos derivados dele
  - Após o período de monitoração, um relatório de atividades é gerado para análise
  - Entretanto, um dos grandes problemas da análise dinâmica é que a interpretação dos relatórios é deixada a cargo do usuário que submeteu o exemplar
  - Limitações das técnicas comumente utilizadas para interceptar as chamadas de sistema ou instrumentar o ambiente da sandbox podem levar à evasão da análise por exemplares de malware modernos
  - Alguns exemplares de malware terem por característica a mutação de seus comportamentos durante execuções distintas, faz com que o relatório gerado por um sistema possa ser diferente do relatório gerado por outro

Técnica funcional da caixa preta

- Sinônimos: orientado a dado / orientado a entrada e saída
  - Concentram-se nos requisitos funcionais do software
    - Os testes são gerados através do estudo de suas entradas e saídas
    - Avalia o comportamento externo do componente de software, sem se considerar o comportamento interno do mesmo
    - Quanto mais entradas são fornecidas, mais rico será o teste
    - Se preocupam com a função que o programa desempenha sem se preocupar com como a função foi implementada
      - Externo: SIM
      - Interno: NÃO

Técnica funcional da caixa preta

- Perspectiva interna do sistema é desconsiderada, sendo testadas e mensuradas somente as interfaces do sistema
- Procura descobrir funções incorretas ou ausentes, erros de interface ou estruturas de dados, erros de desempenho e de inicialização e término
- Essa técnica é aplicável a todas as fases de teste de software

Técnica da caixa cinza

- É um mesclado do uso das técnicas de caixa-preta E de caixa-branca
- Acesso a estruturas de dados e algoritmos do componente a fim de desenvolver os casos de teste, que são executados como na técnica da caixa-preta
- A caixa-cinza pode incluir também o uso de engenharia reversa para determinar por exemplo os limites superiores e inferiores das classes, além de mensagens de erro

Análise Estática x Dinâmica

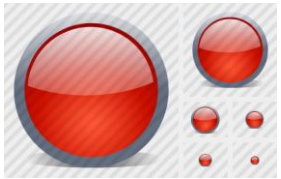
- Debbugers → Análise on-line → Comportamento
- Disassembler → Análise off-line → Código

Análise Post-mortem

- É o estudo do comportamento de um programa examinando os efeitos após a execução
- É na maioria das vezes a única ferramenta disponível após um incidente
- A desvantagem dessa análise é que as informações desaparecem ao longo do tempo porque o comportamento normal do sistema apaga os vestígios
- Ex.
  - Alterações no conteúdo de arquivos
  - Alterações em datas
  - Dados ainda existentes na memória
  - Informações registradas em máquinas externas

RED POINTING

- Método mais fácil e rápido de fazer engenharia reversa de um código
- Exame do código em busca dos pontos fracos óbvios
- Depois de identificadas, força-se a execução do programa de tal forma que essas áreas marcadas sejam percorridas no fluxo de execução

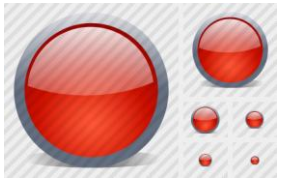


RED POINTING

- Para gerar o red-pointing precisaremos de:

1. Local desprotegido com chamada de sistema potencialmente vulnerável
2. Dados fornecidos pelo usuário que fluem e são processados neste local.

- Desvantagens
  - Deixa escapar quase tudo, exceto os bugs mais triviais



Anti Análise

– Virtualização

- A detecção de ambiente virtualizado também é simples, com apenas uma instrução assembly que, mesmo executada em um nível de baixo privilégio, retorna informações internas sobre o sistema operacional presente no guest
- Tais informações identificam o ambiente virtualizado com base nas diferenças entre estes e sistemas reais

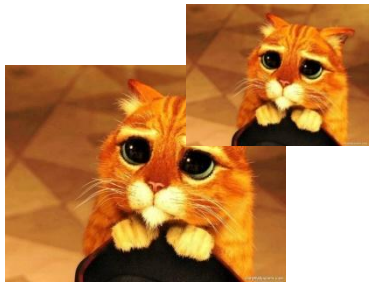
Anti Análise

– Emulação

- Um dos modos utilizados para realizar tal verificação é por meio de bugs conhecidos em processadores de determinadas arquiteturas que fazem com que certas instruções não se comportem como esperado
- Se esta instrução for executada no emulador e este não estiver preparado para apresentar o mesmo comportamento de um processador real, o malware irá perceber essa diferença, podendo parar ou modificar a sua execução

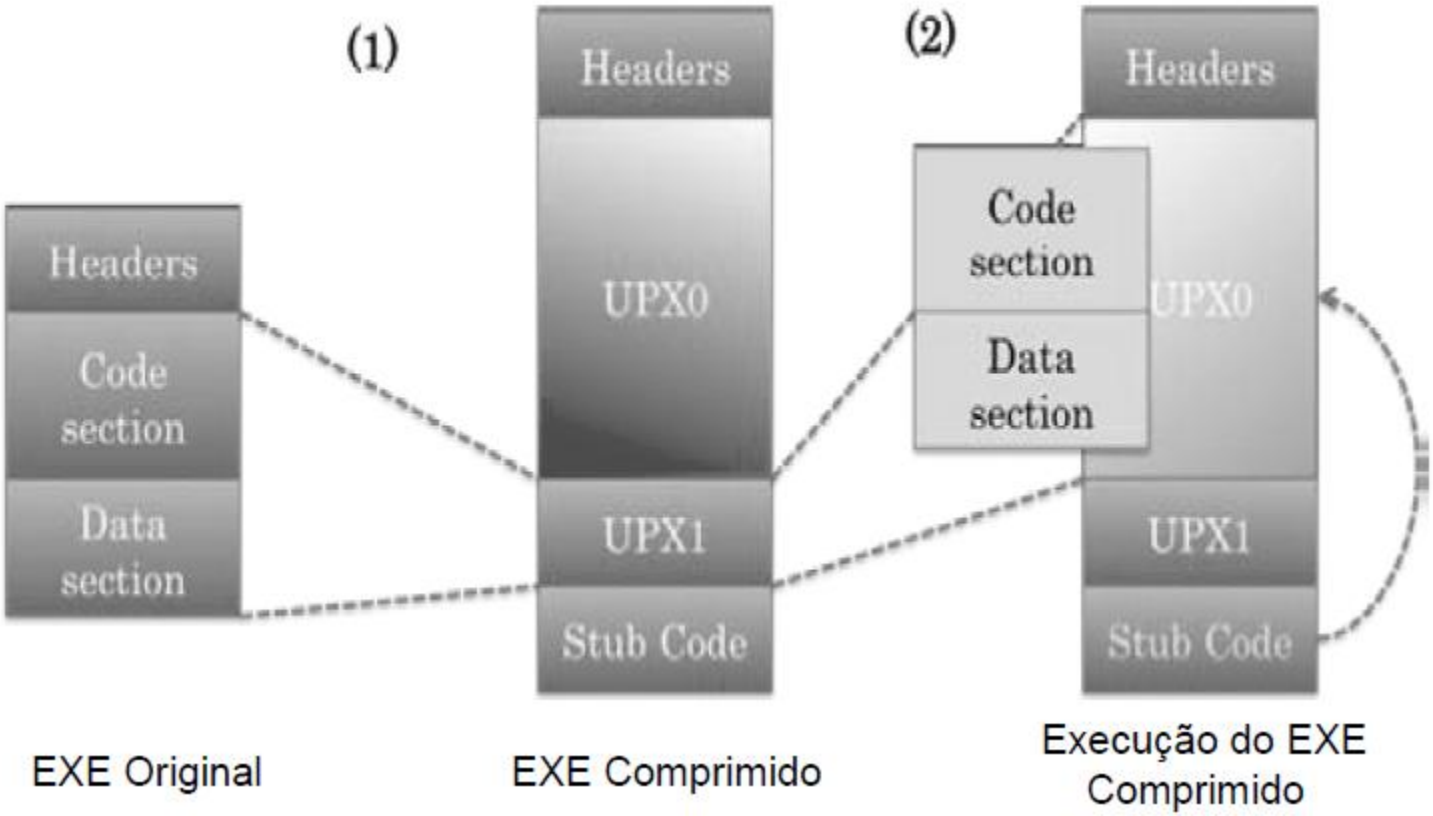
Contornando a Anti Análise

- Para contornar as técnicas de anti análise, existem meios de detectar que um malware verifica se está em ambiente emulado, ou mesmo de modificar alguns valores presentes no ambiente virtual para tentar disfarçá-lo
- Entretanto, o uso destas técnicas muitas vezes é insuficiente e o malware ainda pode detectar que está sendo executado em ambiente emulado/virtual



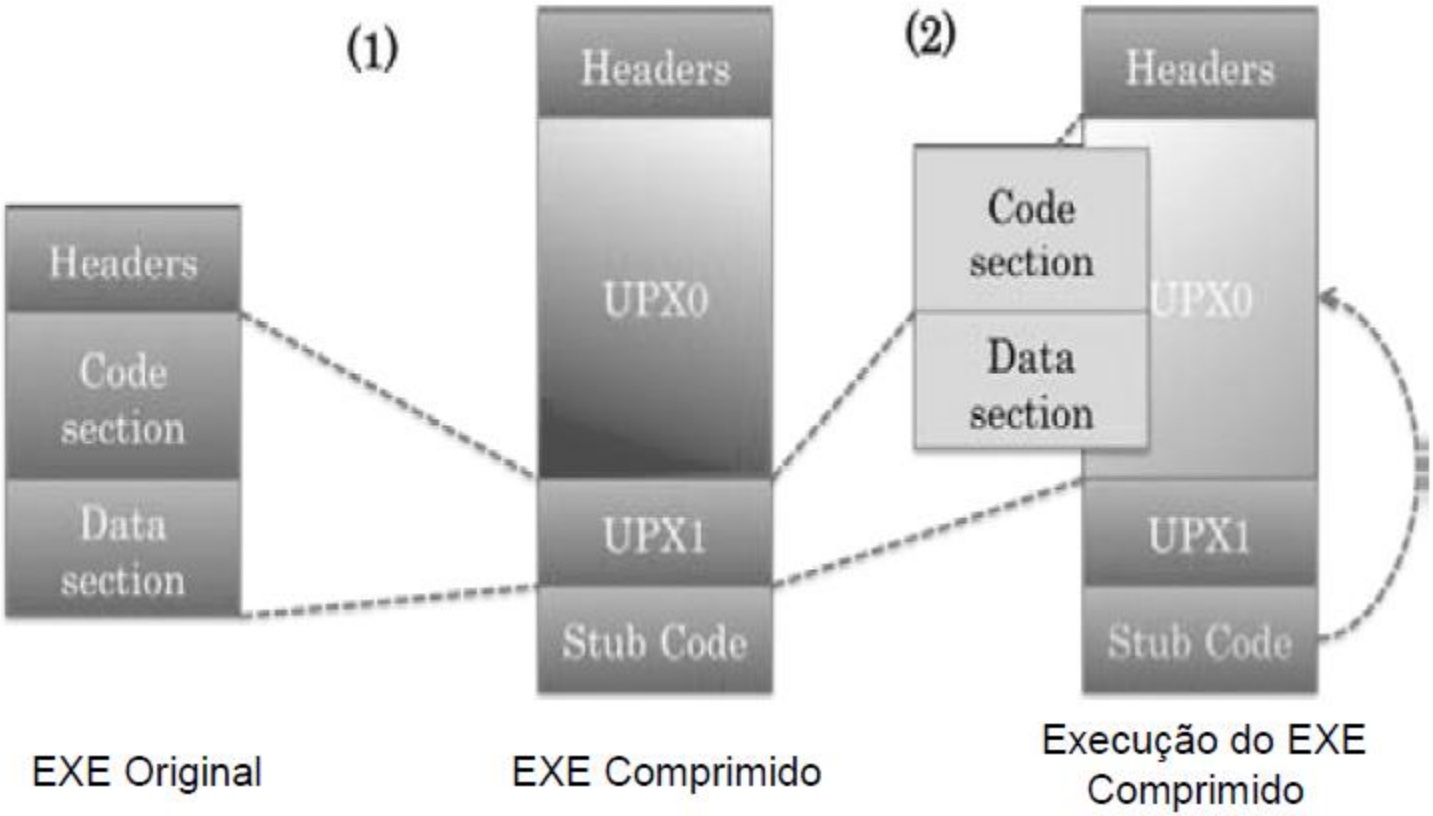
Compactação de Executáveis

- Não se trata da aplicação de uma rotina zip sobre um executável de forma simplista
- Compressão executável é qualquer meio de comprimir um executável e combinar os dados comprimidos com código de descompressão em um único executável
  - Auto-extraível
- Quando este executável compactado é executado, o código de descompressão recria o código original do código comprimido antes de executá-lo



Compactação de Executáveis

- Pode ocorrer sobre um único arquivo ou vários, entretanto ocorre de forma transparente para o usuário
- Não limita a binários, é extensível a scripts, cuja taxa de compactação é maior
- Adiciona um pequeno módulo escrito em assembly no início do arquivo. Esse módulo é responsável por extrair o conteúdo do arquivo para a memória e passar o controle a ele



Compactação de Executáveis

- Estando comprimidos, os arquivos são bem diferentes dos originais. Editores de recursos não poderão acessar os recursos do programa, pois a estrutura dele será diferente
- Modificações com editores hexadecimais também serão frustradas
  - Alterações de um byte sequer com um editor hexadecimal, pode ser que o programa nem funcione
- Só é admitida a compressão sem perdas. A compressão com perdas danifica o executável

Compactação de Executáveis

- Existem dois tipos de compressão que podem ser aplicadas a scripts
  - Reduzir a redundância no script (removendo comentários, espaço em branco e encurtar os nomes de variáveis e funções). Isto não altera o comportamento do script
  - Comprimir o script original e criar um novo script que contém código de descompressão e dados comprimidos. Isto é semelhante à compressão executável binário
- Compactação otimizada para binários (DLLs, EXEs), nem sempre superam os zips na compressão de PPTS, DOCS e outros arquivos de usuário
  - Compressão é aplicada nos resources (ícones, imagens, forms, etc)
  - O binário mesmo é deixado praticamente intacto

Compactação de Executáveis

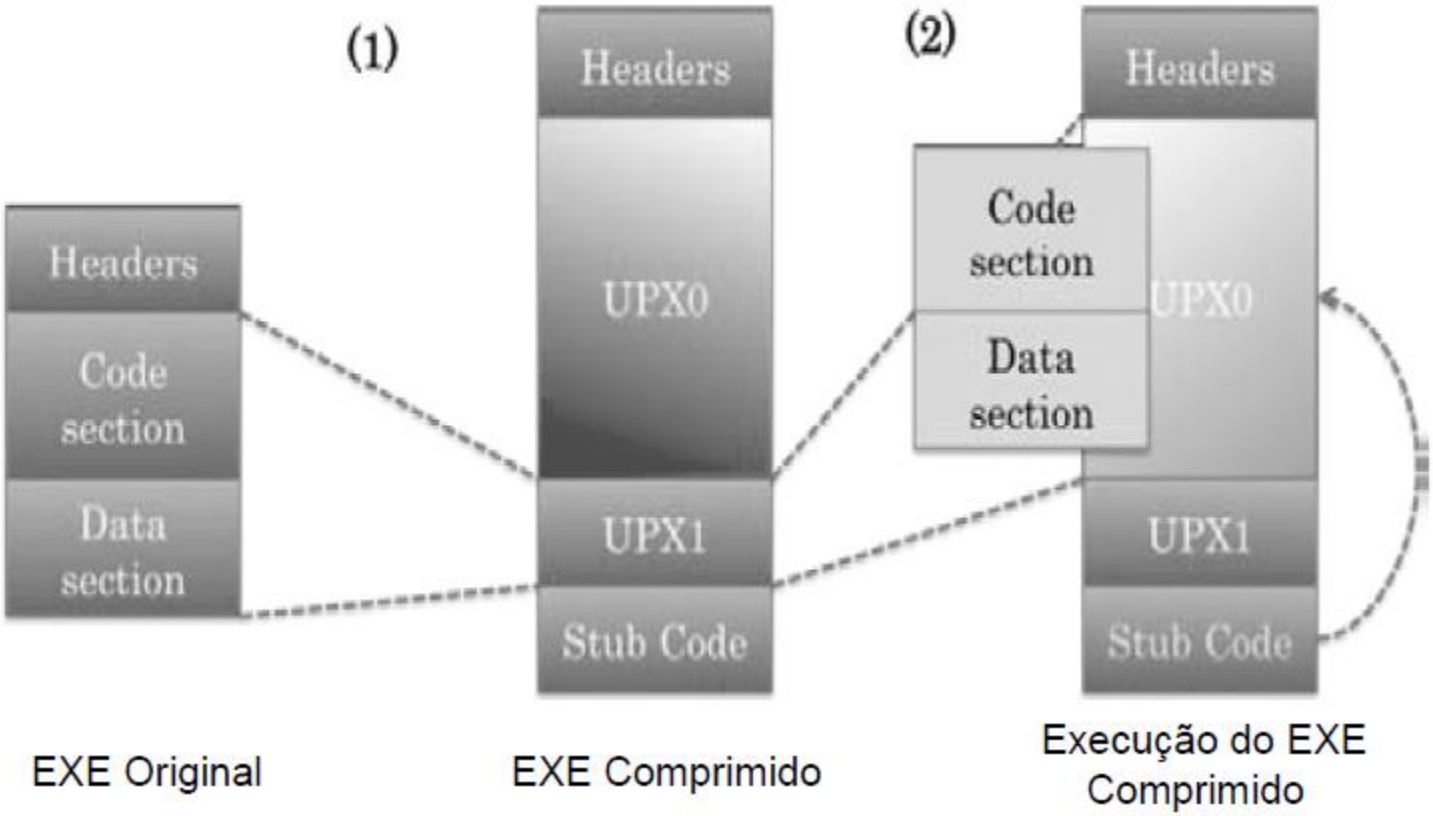
- Compressores de executáveis são projetados especificamente para compactar código executável
  - Obtêm taxas de compressão superiores aos "zip" do mercado
  - 70% em executáveis
- Pode ser combinado com uso da criptografia

Vantagens da Compactação

- Redução de requisitos para armazenamento secundário
- Redução de tempo e largura de banda para distribuição
- Tempo de transferência do disco para a memória menor
- Ideal para dispositivos com pouco requisitos de memória secundária e pouca largura de banda
  - Mas com excelente memória primária

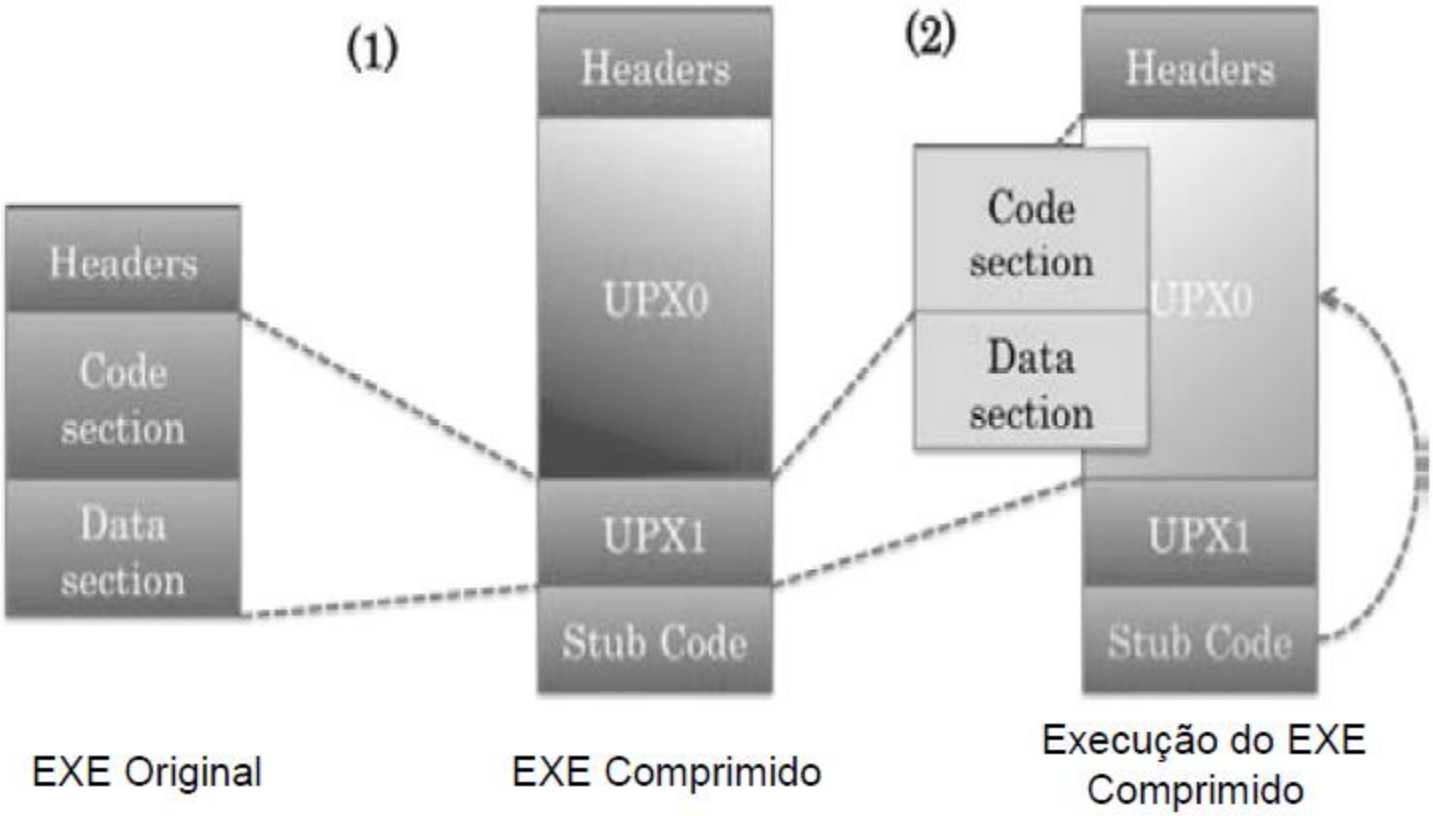
Desvantagens da Compactação

- Requer algum tempo para descomprimir os dados antes da execução
  - Geralmente o armazenamento é o gargalo e não o processamento
  - Não se torna uma desvantagem propriamente dita
  - Quanto maior o executável, maior o tempo de descompressão e consequente percepção por parte do usuário



Desvantagens da Compactação

- Quando um executável comprimido é chamado à memória, um espaço adicional é reservado para acomodar o conteúdo descomprimido
  - Memória para o executável compactado
  - Memória para o executável descompactado
  - Memória para a rotina de descompressão
  - Memória para a rotina de compressão



Desvantagens da Compactação

- No processo de paginação existem várias possibilidades
  - Fazer o swap out somente do executável descompactado, o que facilita o processo de swap in, executável compactado continuará a ocupar espaço na RAM
  - Fazer o swap out do executável compactado descartando o descompactado, apesar de liberar mais RAM, encarece o processo de swap in por necessitar de mais descompactação
  - Alguns executáveis requerem ainda que, após a descompressão, uma cópia descomprimida seja lançada em disco independentemente da necessidade de swap out imediata

Desvantagens da Compactação

- Quando o mesmo executável é carregado mais de uma vez para a memória, existirão 2 instâncias, impedindo a reutilização dos blocos
- Em algumas situações a ligação dinâmica é prejudicada, permitindo que a ligação estática permaneça intacta
- Executáveis compactados são comumente identificados pelos antivírus como código malicioso, pois possuem semelhanças com algumas categorias de vírus - falsos positivos - packers

Desvantagens da Compactação

- Alguns poucos programas apresentam problemas depois de comprimidos. Normalmente são programas que fazem verificações ou acessam a si mesmo
- Alguns programas nem rodam depois de comprimidos. Outros dão mensagens de erros de violação de acesso à memória, ou alguma coisa fica inconforme.

Vantagem ou Desvantagem

- Frequentemente usado para impedir a engenharia reversa ou ofuscar o conteúdo do arquivo executável
  - Impede a detecção correta de código malicioso
- Previne desmontagens diretas
- Modifica assinaturas
- Mascara strings
- Não elimina a engenharia reversa, mas torna o processo mais custoso

Alguns Compactadores

- UPX
  - Gratuito, aberto, linux, windows
- ASPack
  - Windows, fechado, pago
- Petite
  - Windows, fechado, free
- CUP386
- UNP
- Armadillo
- ASPROTECT
- TELOCK
- PELOCK
- eXPressor
- MPRESS
- Enigma
- Cexe
- PKLite
- Exepack.NET
- Themida

Ferramentas case

- Computer Aided Software Engineering - Engenharia de software auxiliada por computador
- Ferramentas que oferecem um conjunto de serviços, fortemente relacionados, para apoiar uma ou mais atividades do processo de desenvolvimento de software
- Abrangem uma larga faixa de diferentes tipos de programas que são usados para apoiar as atividades do processo de software
  - Apoiam a evolução do software em cada etapa do seu ciclo de vida.
  - Apoiam a execução de atividades do desenvolvimento do software de forma automatizada
  - Em alguns casos, implementam um ambiente relativamente refinado no qual várias atividades de especificação ou codificação são apoiadas por recursos computacionais

Ferramentas case

- Podem também incluir um gerador de código que gera automaticamente o código-fonte com base no modelo do sistema
- Sua construção também demanda a utilização das técnicas de desenvolvimento de software – afinal ela é um software
- Objetivos
  - Melhoria da qualidade de software
  - Aumento da produtividade no processo de software

Ferramentas case - vantagens

- Qualidade no produto final
- Produtividade
  - Os aprimoramentos concedidos pelas ferramentas CASE é na ordem de 40%
- Agilizar o tempo para tomada de decisão
- Menor quantidade de códigos de programação
- Melhoria e redução de custos na manutenção
- Agilidade no retrabalho do sw

Ferramentas case - desvantagens

- Incompatibilidade de ferramentas
- Custo alto por licença
- Treinamento para utilização
- O apoio fornecido pelas ferramentas CASE não revolucionou a engenharia de software, como haviam previsto seus proponentes originais, principalmente devido a duas limitações
  - A engenharia de software é essencialmente uma atividade de projeto, baseada no pensamento criativo
  - Por ser uma atividade em equipe onde as pessoas passam muito tempo interagindo

Ferramentas case - automatizações

- Desenvolvimento dos modelos gráficos do sistema
  - Um dos componentes indispensáveis de uma ferramenta CASE é a modelagem visual, ou seja, a possibilidade de representar, através de modelos gráficos, o que está sendo definido. No nosso caso, análise orientada a objetos através da UML

Ferramentas case - automatizações

- Geração de interfaces com o usuário com base em uma descrição
- Debugging de um programa
- Tradução automática através de programas (transliteração)

Ferramentas case - espécies

- Estáticas
  - usam um código-fonte de programa como entrada, analisam e extraem a arquitetura do programa, a estrutura de controle, o fluxo lógico, a estrutura de dados e o fluxo de dados

Ferramentas case - espécies

- Estática
  - Ex:JUDE (astah)
    - Gera código fonte a partir dos diagramas de classe
    - A partir do código fonte criar os modelos

Ferramentas case - espécies

- Dinâmica
  - Monitoram o software quanto a sua execução e usam as informações obtidas durante a monitoração para construir um modelo comportamental do programa
  - Interagem com um sistema em execução, verificando a cobertura dos caminhos, assertivas de teste sobre o valor de variáveis específicas
  - Simulam a função de hardware, ou de outros equipamentos externos
  - Ex: OllyDbg, IDA Pro Disassembler

Ferramentas CASE usadas na engenharia reversa

- Java Optimize and Decompile Environment (JODE)
- JAD - the fast JAva Decompiler
- JReversePro - Java Decompiler
- Win32 Program Disassembler
- IDA Pro Disassembler
- Ollydbg
- DeDe

Técnicas Anti Engenharia Reversa

- É impossível criar-se um código imune ao cracking
  - mas é possível dificultar a vida do cracker
  - O que se faz é esconder, mascarar, enganar e obstruir o programa
  - penalidade sofrida é a queda de performance em tempo de execução

Técnicas Anti Engenharia Reversa

- **Solução 1**: é a combinação de técnicas de forma que as reações colaterais indesejadas sejam minimizadas
  - Lentidão
  - Aumento do espaço ocupado
  - Às vezes não
- **Solução 2**: Aplicação das técnicas anticracking em pontos do programa que necessitam mais de segurança do que eficiência
  - Checagens de serial, registro, validade de instalação etc
  - Usuário não percebe a queda de eficiência relativa para a funcionalidade que foi protegida

Técnicas Anti Engenharia Reversa - Ofuscação

- Usuários mal intencionados podem, através de uma análise profunda do aplicativo, reconhecer o código fonte e manipula-lo a fim de retirar informações para benefício próprio.
- Técnicas de segurança, como a ofuscação de código podem dificultar esse trabalho e tornar o software mais confiável e seguro. Quando disassemblamos um binário para crackearmos, muitas vezes nos deparamos com um código que parece não fazer sentido
  - Isso ocorre por que o programa possivelmente foi implantado com alguma proteção contra engenharia reversa.

Técnicas Anti Engenharia Reversa - Ofuscação

- São muitos os casos em que se deseja criar softwares com proteção contra reversão
  - Criadores de vírus
  - Empresas colocando camadas adicionais de segurança
- Algumas técnicas usadas para tornar o código compilado mais imune à ação de reversers e as consequências adversas de sua implementação
  - Perda de desempenho
  - Aumento da complexidade do sistema

Técnicas Anti Engenharia Reversa - Ofuscação

- São várias as abordagens que podemos tomar para proteger um código
- Normalmente as aplicações utilizam uma combinação das técnicas que veremos para minimizar a produtividade do trabalho dos crackers.

Técnicas Anti Engenharia Reversa - Ofuscação

- Programas de código aberto não necessitam de ofuscamento
  - O código já é aberto
  - Sistemas abertos costumam ter eficiência computacional superior se compararmos com programas com fins semelhantes
  - inserção de meios que impedem a reversão no sistema proprietário

Técnicas Anti Engenharia Reversa - Ofuscação

- Encriptação e ofuscação são técnicas utilizadas para reduzir a vulnerabilidade de crackear programas
- Consiste em modificar o layout do programa, a lógica e os dados de forma que reorganize o código, tornando-o menos legível, mas mantendo a funcionalidade para o usuário final

Técnicas Anti Engenharia Reversa - Ofuscação

- Repetição de nomes de identificadores
  - Ex: Em Java colocam as classes, métodos e variáveis com mesmo nomes
- Cifragem de strings
  - Strings, que teoricamente ficariam intactas, são modificadas
- Mudanças no fluxo de execução
  - Prejudica a análise estática do código
- Ofuscação pura diferencia-se da Encriptação basicamente pela dependência da chave

REFATORAÇÃO

- É o processo de modificar um sistema de software para melhorar a estrutura interna do código sem alterar seu comportamento externo.

PATCH

- Termo da língua inglesa que significa remendo
- É um programa de computador criado para atualizar ou corrigir um software
- Patching - É o ato de alterar determinado programa e tornar a alteração parte do próprio programa
- Promove alterações na estrutura e fluxo lógico de execução.

Forense para Concursos

Bateria de Questões de Aprendizagem

1. Acerca dos conceitos da engenharia reversa, julgue os itens subsecutivos.

[63] A depuração de programas utiliza métodos de teste e análise para tentar entender o software. Esses métodos são classificados como caixa-branca (white box) e caixa-preta (black box). Para se conhecer o código e seu comportamento, o teste caixa-branca é menos eficiente que o teste caixa-preta, embora seja mais fácil de ser implementado.

[64] Red pointing é o método mais rápido para se realizar engenharia reversa em um código. Para criar um red pointing em um código alvo, é suficiente identificar no programa os locais potencialmente vulneráveis, que fazem chamada ao sistema operacional, e detectar os dados fornecidos pelo usuário, que são processados nesse local.

[65] A engenharia reversa permite conhecer a estrutura do programa e sua lógica e, com base nessas informações, alterar a estrutura do programa, afetando diretamente o fluxo lógico. Essa atividade é conhecida como patching.

1. Acerca dos conceitos da engenharia reversa, julgue os itens subsecutivos.

~~[63] A depuração de programas utiliza métodos de teste e análise para tentar entender o software. Esses métodos são classificados como caixa-branca (white box) e caixa-preta (black box). Para se conhecer o código e seu comportamento, o teste caixa-branca é menos eficiente que o teste caixa-preta, embora seja mais fácil de ser implementado.~~

[64] Red pointing é o método mais rápido para se realizar engenharia reversa em um código. Para criar um red pointing em um código alvo, é suficiente identificar no programa os locais potencialmente vulneráveis, que fazem chamada ao sistema operacional, e detectar os dados fornecidos pelo usuário, que são processados nesse local.

[65] A engenharia reversa permite conhecer a estrutura do programa e sua lógica e, com base nessas informações, alterar a estrutura do programa, afetando diretamente o fluxo lógico. Essa atividade é conhecida como patching.

2. Considerando que todo desenvolvimento de software pode ser entendido como um ciclo de solução de problemas, julgue os itens a seguir, relativos a gerenciamento de processos de negócio.

[114] A engenharia reversa consiste no processo de levantamento de requisitos de um sistema sem documentação. Na engenharia reversa, a técnica caixa branca é empregada para observar os inputs e outputs do sistema.

2. Considerando que todo desenvolvimento de software pode ser entendido como um ciclo de solução de problemas, julgue os itens a seguir, relativos a gerenciamento de processos de negócio.

~~[114] A engenharia reversa consiste no processo de levantamento de requisitos de um sistema sem documentação. Na engenharia reversa, a técnica caixa branca é empregada para observar os inputs e outputs do sistema.~~

3. Com relação à fase de manutenção, julgue os itens subsequentes.

[80] Entre os problemas comuns na fase de manutenção, podem-se citar a baixa produtividade e o alto custo, além de problemas técnicos, como documentação desatualizada e dificuldade de se alterarem sistemas que foram projetados sem a preocupação com a sua manutenibilidade.

[81] A engenharia reversa se propõe a gerar uma nova especificação ou um novo projeto de um sistema existente, aplicando-se os conceitos de reengenharia, para, depois, se aplicar a engenharia direta e gerar um novo sistema.

[82] Os sistemas de software estão sujeitos a manutenção, pois, dificilmente, as características que o definem não sofrem modificações durante a sua vida útil. Quanto mais dependentes do mundo real forem os requisitos de um sistema, maior será a probabilidade de esse sistema vir a ser modificado.

[83] Um sistema pode sofrer quatro tipos de manutenção: corretiva, adaptativa ou evolutiva, perfectiva e preventiva. Entre esses tipos, apenas a manutenção corretiva está associada a um defeito do sistema.

3. Com relação à fase de manutenção, julgue os itens subsequentes.

[80] Entre os problemas comuns na fase de manutenção, podem-se citar a baixa produtividade e o alto custo, além de problemas técnicos, como documentação desatualizada e dificuldade de se alterarem sistemas que foram projetados sem a preocupação com a sua manutenibilidade.

~~[81] A engenharia reversa se propõe a gerar uma nova especificação ou um novo projeto de um sistema existente, aplicando-se os conceitos de reengenharia, para, depois, se aplicar a engenharia direta e gerar um novo sistema.~~

[82] Os sistemas de software estão sujeitos a manutenção, pois, dificilmente, as características que o definem não sofrem modificações durante a sua vida útil. Quanto mais dependentes do mundo real forem os requisitos de um sistema, maior será a probabilidade de esse sistema vir a ser modificado.

~~[83] Um sistema pode sofrer quatro tipos de manutenção: corretiva, adaptativa ou evolutiva, perfectiva e preventiva. Entre esses tipos, apenas a manutenção corretiva está associada a um defeito do sistema.~~

4. Analisando as sentenças seguintes, relativas a desenvolvimento de software,

I. Compiladores são programas que geram código executável por um computador, a partir de um conjunto de instruções expressos em uma linguagem de programação.

II. Ligadores são programas que entrelaçam os códigos de vários programas interdependentes, de forma a codificar e dificultar engenharia reversa do código.

III. Tanto os compiladores quanto os interpretadores geram código executável por um computador. A principal diferença entre compilador e interpretador é o momento em que esse código é gerado.

IV. Montadores são uma espécie de compiladores de baixo nível, isto é, programas que transformam um programa escrito em uma linguagem de programação de baixo nível, como Assembly, em um código binário executável.

verifica-se que

A. somente I e II estão corretas.
B. somente a II está correta.
C. somente I, III e IV estão corretas.
D. somente I e III estão corretas.
E. todas estão corretas.

4. Analisando as sentenças seguintes, relativas a desenvolvimento de software,

I. Compiladores são programas que geram código executável por um computador, a partir de um conjunto de instruções expressos em uma linguagem de programação.

II. Ligadores são programas que entrelaçam os códigos de vários programas interdependentes, de forma a codificar e dificultar engenharia reversa do código.

III. Tanto os compiladores quanto os interpretadores geram código executável por um computador. A principal diferença entre compilador e interpretador é o momento em que esse código é gerado.

IV. Montadores são uma espécie de compiladores de baixo nível, isto é, programas que transformam um programa escrito em uma linguagem de programação de baixo nível, como Assembly, em um código binário executável.

verifica-se que

A. somente I e II estão corretas.
B. somente a II está correta.
C. somente I, III e IV estão corretas.
D. somente I e III estão corretas.
E. todas estão corretas.

5. Identifique as alternativas corretas a respeito de engenharia reversa.

1. Descompiladores são usados para obter o código fonte de um software a partir de seu código binário.
2. Ofuscadores de código efetuam a cifragem de códigos binários de programas com o intuito de impedir a sua descompilação.
3. Através de técnicas de engenharia reversa, é possível obter diagramas UML de um programa a partir de seu código fonte.
4. Descompilação de código e esteganografia são duas técnicas frequentemente usadas para realizar a engenharia reversa de sistemas computacionais.

Assinale a alternativa que indica todas as afirmativas corretas.

A. São corretas apenas as afirmativas 1 e 3.
B. São corretas apenas as afirmativas 1 e 4.
C. São corretas apenas as afirmativas 2 e 3.
D. São corretas apenas as afirmativas 1, 2 e 4.
E. São corretas apenas as afirmativas 2, 3 e 4.

5. Identifique as alternativas corretas a respeito de engenharia reversa.

1. Descompiladores são usados para obter o código fonte de um software a partir de seu código binário.
2. Ofuscadores de código efetuam a cifragem de códigos binários de programas com o intuito de impedir a sua descompilação.
3. Através de técnicas de engenharia reversa, é possível obter diagramas UML de um programa a partir de seu código fonte.
4. Descompilação de código e esteganografia são duas técnicas frequentemente usadas para realizar a engenharia reversa de sistemas computacionais.

Assinale a alternativa que indica todas as afirmativas corretas.

A. São corretas apenas as afirmativas 1 e 3.

B. São corretas apenas as afirmativas 1 e 4.

C. São corretas apenas as afirmativas 2 e 3.

D. São corretas apenas as afirmativas 1, 2 e 4.

E. São corretas apenas as afirmativas 2, 3 e 4.

6. A respeito de depuração de programas e algoritmos, julgue os itens subsequentes.

[51] Para a depuração de programas em sistema operacional Windows, é necessário que o espaço de memória RAM esteja paginado, além do uso de uma máquina virtual conectada diretamente ao controle da CPU.

[52] É possível depurar vários processos ao mesmo tempo, mesmo que estejam escritos em linguagens diferentes.

[53] Em um processo de depuração tradicional, qualquer ferramenta de depuração pode interpretar o código de máquina e fazer a engenharia reversa das funções de um programa.

[54] Normalmente, no processo de depuração de um código em linguagem C, saber em que ponto do programa aconteceu um erro é suficiente para se determinar qual é o caminho até a causa do erro, independentemente da sequência de funções chamadas.

[55] Um processo de depuração pode ser dividido em on-line e off-line, e se diferenciam pelo tipo de interação com a execução do programa. A depuração on-line oferece interação direta com a aplicação, enquanto a off-line interage com um arquivo de monitoração gravado durante a execução da aplicação.

6. A respeito de depuração de programas e algoritmos, julgue os itens subsequentes.

~~[51] Para a depuração de programas em sistema operacional Windows, é necessário que o espaço de memória RAM esteja paginado, além do uso de uma máquina virtual conectada diretamente ao controle da CPU.~~

[52] É possível depurar vários processos ao mesmo tempo, mesmo que estejam escritos em linguagens diferentes.

~~[53] Em um processo de depuração tradicional, qualquer ferramenta de depuração pode interpretar o código de máquina e fazer a engenharia reversa das funções de um programa.~~

~~[54] Normalmente, no processo de depuração de um código em linguagem C, saber em que ponto do programa aconteceu um erro é suficiente para se determinar qual é o caminho até a causa do erro, independentemente da sequência de funções chamadas.~~

[55] Um processo de depuração pode ser dividido em on-line e off-line, e se diferenciam pelo tipo de interação com a execução do programa. A depuração on-line oferece interação direta com a aplicação, enquanto a off-line interage com um arquivo de monitoração gravado durante a execução da aplicação.

7. A ENGENHARIA REVERSA constitui uma técnica usada para tentar obter o código fonte do programa a partir do arquivo já compilado.

A esse respeito, analise as afirmativas abaixo, sobre ferramentas disponíveis.

I. Java Optimize and Decompile Environment (JODE) é um pacote java contendo um descompilador e um otimizador e ofuscador para java. Este package é livremente disponível sob licença GNU GPL.

II. JAD - the fast JAva Decompiler é um descompilador Java, um programa que lê um ou mais arquivos class e os converte em fontes Java. JAD não usa o Java runtime para o seu funcionamento, sendo gratuito para uso nãocomercial.

III. JReversePro - Java Decompiler é um Java Descompilador / Disassembler escrito inteiramente em Java, sendo um utilitário de engenharia reversa sob licença GNU GPL.

Assinale:

A. se somente a afirmativa II estiver correta.
B. se somente as afirmativas I e II estiverem corretas.
C. se somente as afirmativas I e III estiverem corretas.
D. se somente as afirmativas II e III estiverem corretas.
E. se todas as afirmativas estiverem corretas.

7. A ENGENHARIA REVERSA constitui uma técnica usada para tentar obter o código fonte do programa a partir do arquivo já compilado.

A esse respeito, analise as afirmativas abaixo, sobre ferramentas disponíveis.

I. Java Optimize and Decompile Environment (JODE) é um pacote java contendo um descompilador e um otimizador e ofuscador para java. Este package é livremente disponível sob licença GNU GPL.

II. JAD - the fast JAva Decompiler é um descompilador Java, um programa que lê um ou mais arquivos class e os converte em fontes Java. JAD não usa o Java runtime para o seu funcionamento, sendo gratuito para uso nãocomercial.

III. JReversePro - Java Decompiler é um Java Descompilador / Disassembler escrito inteiramente em Java, sendo um utilitário de engenharia reversa sob licença GNU GPL.

Assinale:

A. se somente a afirmativa II estiver correta.
B. se somente as afirmativas I e II estiverem corretas.
C. se somente as afirmativas I e III estiverem corretas.
D. se somente as afirmativas II e III estiverem corretas.
E. se todas as afirmativas estiverem corretas.

8. Acerca da reengenharia e da engenharia reversa, julgue os seguintes itens.

I - A engenharia reversa visa descobrir os princípios tecnológicos de um sistema de software via análise da sua estrutura, função e operação. Para realizar a engenharia reversa, examina-se o que um sistema faz e como faz.

II - Decompiladores podem auxiliar na engenharia reversa. Um decompilador traduz programas executáveis em código-fonte. Por exemplo, há decompiladores que convertem arquivos class em arquivos fonte Java.

III - Há compiladores que, visando dificultar a engenharia reversa, produzem códigos difíceis de decompilar. A técnica code obfuscation combate a ação desses compiladores, pois produz fontes fáceis de entender a partir de códigos executáveis.

IV - Um depurador pode ser usado para facilitar a engenharia reversa de um software. Esse tipo de ferramenta pode prover facilidades para controlar a execução, ler ou escrever em posições da memória.

V - Padrões de refatoração podem ser usados na reengenharia de um software. Em um programa Java, mover atributos entre classes e alterar parâmetros de métodos são exemplos de padrões de refatoração. Há ambientes de desenvolvimento (IDEs) que facilitam a refatoração de código-fonte Java.

A quantidade de itens certos é igual a

A. 2.

B. 3.

C. 4.

D. 5.

8. Acerca da reengenharia e da engenharia reversa, julgue os seguintes itens.

I - A engenharia reversa visa descobrir os princípios tecnológicos de um sistema de software via análise da sua estrutura, função e operação. Para realizar a engenharia reversa, examina-se o que um sistema faz e como faz.

II - Decompiladores podem auxiliar na engenharia reversa. Um decompilador traduz programas executáveis em código-fonte. Por exemplo, há decompiladores que convertem arquivos class em arquivos fonte Java.

III - Há compiladores que, visando dificultar a engenharia reversa, produzem códigos difíceis de decompilar. A técnica code obfuscation combate a ação desses compiladores, pois produz fontes fáceis de entender a partir de códigos executáveis.

IV - Um depurador pode ser usado para facilitar a engenharia reversa de um software. Esse tipo de ferramenta pode prover facilidades para controlar a execução, ler ou escrever em posições da memória.

V - Padrões de refatoração podem ser usados na reengenharia de um software. Em um programa Java, mover atributos entre classes e alterar parâmetros de métodos são exemplos de padrões de refatoração. Há ambientes de desenvolvimento (IDEs) que facilitam a refatoração de código-fonte Java.

A quantidade de itens certos é igual a

A. 2.

B. 3.

C. 4.

D. 5.

9. Analise as afirmativas a seguir, relativas à reengenharia de sistemas.

1. A Engenharia Reversa pode ser utilizada para construir a modelagem de programas a partir do seu código-fonte.
2. Os pacotes de dados capturados nas redes de computadores através do uso de sniffers podem ser utilizados para realizar a engenharia reversa do protocolo empregado na troca das mensagens.
3. A Engenharia Reversa pode ser utilizada por crackers no desenvolvimento de malwares, por isso não pode ser utilizada no desenvolvimento dos antídotos destes malwares, dificultando a detecção de vestígios que permitam identificar o seu criador.

Está(ão) *correta(s):*

A. as afirmativas 1 e 2, apenas.
B. a afirmativa 1, apenas.
C. as afirmativas 2 e 3, apenas.
D. as afirmativas 1 e 3, apenas.
E. as afirmativas 1, 2, 3.

9. Analise as afirmativas a seguir, relativas à reengenharia de sistemas.

1. A Engenharia Reversa pode ser utilizada para construir a modelagem de programas a partir do seu código-fonte.
2. Os pacotes de dados capturados nas redes de computadores através do uso de sniffers podem ser utilizados para realizar a engenharia reversa do protocolo empregado na troca das mensagens.
3. A Engenharia Reversa pode ser utilizada por crackers no desenvolvimento de malwares, por isso não pode ser utilizada no desenvolvimento dos antídotos destes malwares, dificultando a detecção de vestígios que permitam identificar o seu criador.

Está(ão) *correta(s):*

A. as afirmativas 1 e 2, apenas.

B. a afirmativa 1, apenas.

C. as afirmativas 2 e 3, apenas.

D. as afirmativas 1 e 3, apenas.

E. as afirmativas 1, 2, 3.

10. Considere as afirmações abaixo:

I - Há ferramentas CASE que fazem engenharia reversa e engenharia direta.

II - Num processo de reengenharia de software não podem ser usadas ferramentas de engenharia direta.

III - Reengenharia de software é o mesmo que Engenharia Reversa de Software.

É(são) correta(s) apenas:

A. I e II
B. III
C. II
D. I

10. Considere as afirmações abaixo:

I - Há ferramentas CASE que fazem engenharia reversa e engenharia direta.

II - Num processo de reengenharia de software não podem ser usadas ferramentas de engenharia direta.

III - Reengenharia de software é o mesmo que Engenharia Reversa de Software.

É(são) correta(s) apenas:

A. I e II
B. III
C. II
D. I

GABARITO

1. E, C, C
2. E
3. C, E, C, E
4. C
5. A
6. E, C, E, E, C
7. E
8. C
9. A
10. D

Forense para Concursos

Bateria de Questões de Aprendizagem

01. Reengenharia e engenharia reversa são ferramentas que devem ser aplicáveis na manutenção de sistemas em abordagem

A. evolutiva.
B. corretiva.
C. preventiva.
D. complementar.
E. alternativa.

01. Reengenharia e engenharia reversa são ferramentas que devem ser aplicáveis na manutenção de sistemas em abordagem

A. evolutiva.
B. corretiva.
C. preventiva.
D. complementar.
E. alternativa.

02. Considere as afirmações abaixo, referentes à Engenharia de Software:

I - Ferramentas de engenharia reversa, em geral, conseguem gerar informações de projeto a partir de código fonte.

II - Quanto mais alto for o nível de abstração da informação gerada por uma ferramenta de engenharia reversa, mais incompleta tende a ser esta informação gerada.

III - Toda ferramenta que faz engenharia direta e também reversa, garante que uma alteração nas informações de projeto será refletida imediatamente no código fonte e vice-versa.

É(são) correta(s) apenas:

A. I
B. I e II
C. I e III
D. II e III

02. Considere as afirmações abaixo, referentes à Engenharia de Software:

I - Ferramentas de engenharia reversa, em geral, conseguem gerar informações de projeto a partir de código fonte.

II - Quanto mais alto for o nível de abstração da informação gerada por uma ferramenta de engenharia reversa, mais incompleta tende a ser esta informação gerada.

III - Toda ferramenta que faz engenharia direta e também reversa, garante que uma alteração nas informações de projeto será refletida imediatamente no código fonte e vice-versa.

É(são) correta(s) apenas:

A. I

B. I e II

C. I e III

D. II e III

03. A reengenharia de sistemas é um importante processo na área de engenharia de software, em função da larga aplicação do software nas atividades socioeconômicas e da conseqüente necessidade de reconstruir o software, ou porque deixou de atender ao conjunto de necessidades ou porque apresenta algum problema específico de desempenho, segurança, robustez etc.

Acerca dos processos de reengenharia de sistemas e de técnicas a eles associadas, julgue os seguintes itens.

1. Em geral, durante a vida útil dos sistemas computacionais, constata- se que a ocorrência de codificações é inevitável, de modo que é necessário desenvolver mecanismos para avaliar, controlar e realizar modificações.
2. Apenas recentemente o problema da manutenção de software tornou- se grave para o setor, visto que anteriormente o software tinha aplicação relativamente restrita.
3. A reengenharia de sistemas consiste de um processo e de uma técnica de aplicação tipicamente rápida, envolvendo poucos recursos e com resultados a curto prazo.
4. A criação de documentação e a manutenção de uma documentação atualizada é uma das atividades que mais consomem recursos na engenharia de software.
5. Constata-se que a engenharia reversa, a partir do código fonte, resulta, em geral, na obtenção de uma boa documentação, um entendimento bastante abrangente das abstrações dos sistemas e a recuperação das informações de concepção dos sistemas.

03. A reengenharia de sistemas é um importante processo na área de engenharia de software, em função da larga aplicação do software nas atividades socioeconômicas e da conseqüente necessidade de reconstruir o software, ou porque deixou de atender ao conjunto de necessidades ou porque apresenta algum problema específico de desempenho, segurança, robustez etc.

Acerca dos processos de reengenharia de sistemas e de técnicas a eles associadas, julgue os seguintes itens.

1. Em geral, durante a vida útil dos sistemas computacionais, constata- se que a ocorrência de codificações é inevitável, de modo que é necessário desenvolver mecanismos para avaliar, controlar e realizar modificações.
2. ~~Apenas recentemente o problema da manutenção de software tornou- se grave para o setor, visto que anteriormente o software tinha aplicação relativamente restrita.~~
3. ~~A reengenharia de sistemas consiste de um processo e de uma técnica de aplicação tipicamente rápida, envolvendo poucos recursos e com resultados a curto prazo.~~
4. A criação de documentação e a manutenção de uma documentação atualizada é uma das atividades que mais consomem recursos na engenharia de software.
5. ~~Constata-se que a engenharia reversa, a partir do código fonte, resulta, em geral, na obtenção de uma boa documentação, um entendimento bastante abrangente das abstrações dos sistemas e a recuperação das informações de concepção dos sistemas.~~

04. Analise as seguintes afirmações relativas a ferramentas CASE para manutenção de software:

I. Uma ferramenta de engenharia reversa estática deve monitorar o software quanto à sua execução e usar as informações obtidas durante a monitoração para construir um modelo comportamental do programa.

II. Uma ferramenta de engenharia reversa executa uma análise pós-desenvolvimento em um programa existente e pode ser classificada como estática e dinâmica.

III. Uma ferramenta de engenharia reversa dinâmica deve monitorar o software quanto à sua execução para extrair a arquitetura do programa.

IV. Uma ferramenta de engenharia reversa estática deve usar o código fonte do programa para extrair a sua arquitetura.

Indique a opção que contenha todas as afirmações verdadeiras.

A. I e II
B. II e III
C. III e IV
D. I e III
E. II e IV

04. Analise as seguintes afirmações relativas a ferramentas CASE para manutenção de software:

I. Uma ferramenta de engenharia reversa estática deve monitorar o software quanto à sua execução e usar as informações obtidas durante a monitoração para construir um modelo comportamental do programa.

II. Uma ferramenta de engenharia reversa executa uma análise pós-desenvolvimento em um programa existente e pode ser classificada como estática e dinâmica.

III. Uma ferramenta de engenharia reversa dinâmica deve monitorar o software quanto à sua execução para extrair a arquitetura do programa.

IV. Uma ferramenta de engenharia reversa estática deve usar o código fonte do programa para extrair a sua arquitetura.

Indique a opção que contenha todas as afirmações verdadeiras.

A. I e II
B. II e III
C. III e IV
D. I e III
E. II e IV

05. Técnicas de reengenharia de sistemas vêm sendo incorporadas sistematicamente à prática corrente da engenharia de *software, em especial devido ao suporte oferecido pelas modernas ferramentas CASE a esse tipo de técnica. Assim, as técnicas de engenharia direta e reversa, de reestruturação de código e de documentação integram-se às técnicas de gestão de configuração, documentação e requisitos, entre outras. Em atividades de auditoria de sistemas, a utilização de técnicas de reengenharia assistida por uma ferramenta CASE pode ser bastante útil para revelar detalhes internos de sistemas existentes, muitas vezes ocultos na documentação disponível, bem como auxiliar na identificação de alterações de configuração, documentação e especificação de requisitos desses sistemas. Acerca da utilização de técnicas de reengenharia assistidas por ferramentas CASE em auditoria de sistemas, julgue os itens a seguir.*

1. Quando a documentação de um programa ou sistema não está disponível, é suficiente realizar uma engenharia reversa automática a partir do código-fonte para descobrir quais são as suas principais funcionalidades e a semântica de suas estruturas de dados internas mais importantes. Entretanto, os detalhes de implementação internos a cada funcionalidade não podem ser revelados ou evidenciados com esse tipo de técnica.
2. A engenharia direta, a partir da documentação e dos modelos existentes em ferramenta CASE, pode ser usada para gerar a estrutura básica do código-fonte correspondente a esses modelos. Tal código, gerado automaticamente, pode ser usado em termos comparativos com o código-fonte do programa que está sendo analisado, com o objetivo de identificar diferenças entre as especificações constantes da documentação e as estruturas realmente implementadas.
3. Alterações maliciosas em programas podem ser detectadas automaticamente com o emprego sistemático de ferramentas adequadas de controle de versão, que mantêm indicadores de integridade do código-fonte e do código executável ou que podem determinar diferenças entre versões anteriores existentes em cópias de segurança e em versões mais novas.
4. Ferramentas de controle de versão, que mantêm controle de alterações embasado nos registros de datas de modificação e exclusão de arquivos integrados e mantidos pelo sistema operacional, geram informações e revelam, sem equívocos, a ocorrência de alterações em qualquer arquivo do projeto, mesmo que a natureza da alteração não possa ser claramente identificada.
5. Registros de log gerados pelas ferramentas CASE, quando de seu uso sistemático no desenvolvimento de sistemas, auxiliam na descoberta de trilhas de auditoria de modificações nesses sistemas.

05. Técnicas de reengenharia de sistemas vêm sendo incorporadas sistematicamente à prática corrente da engenharia de *software, em especial devido ao suporte oferecido pelas modernas ferramentas CASE a esse tipo de técnica. Assim, as técnicas de engenharia direta e reversa, de reestruturação de código e de documentação integram-se às técnicas de gestão de configuração, documentação e requisitos, entre outras. Em atividades de auditoria de sistemas, a utilização de técnicas de reengenharia assistida por uma ferramenta CASE pode ser bastante útil para revelar detalhes internos de sistemas existentes, muitas vezes ocultos na documentação disponível, bem como auxiliar na identificação de alterações de configuração, documentação e especificação de requisitos desses sistemas. Acerca da utilização de técnicas de reengenharia assistidas por ferramentas CASE em auditoria de sistemas, julgue os itens a seguir.*

1. ~~Quando a documentação de um programa ou sistema não está disponível, é suficiente realizar uma engenharia reversa automática a partir do código-fonte para descobrir quais são as suas principais funcionalidades e a semântica de suas estruturas de dados internas mais importantes. Entretanto, os detalhes de implementação internos a cada funcionalidade não podem ser revelados ou evidenciados com esse tipo de técnica.~~

2. A engenharia direta, a partir da documentação e dos modelos existentes em ferramenta CASE, pode ser usada para gerar a estrutura básica do código-fonte correspondente a esses modelos. Tal código, gerado automaticamente, pode ser usado em termos comparativos com o código-fonte do programa que está sendo analisado, com o objetivo de identificar diferenças entre as especificações constantes da documentação e as estruturas realmente implementadas.

3. Alterações maliciosas em programas podem ser detectadas automaticamente com o emprego sistemático de ferramentas adequadas de controle de versão, que mantêm indicadores de integridade do código-fonte e do código executável ou que podem determinar diferenças entre versões anteriores existentes em cópias de segurança e em versões mais novas.

4. ~~Ferramentas de controle de versão, que mantêm controle de alterações embasado nos registros de datas de modificação e exclusão de arquivos integrados e mantidos pelo sistema operacional, geram informações e revelam, sem equívocos, a ocorrência de alterações em qualquer arquivo do projeto, mesmo que a natureza da alteração não possa ser claramente identificada.~~

5. Registros de log gerados pelas ferramentas CASE, quando de seu uso sistemático no desenvolvimento de sistemas, auxiliam na descoberta de trilhas de auditoria de modificações nesses sistemas.

06. O desenvolvimento de novos sistemas a partir da reengenharia de sistemas de informação e com a utilização de técnicas e de método, adequados pode significar uma grande economia no processo de transferência do conhecimento acumulado nos sistemas legados para os sistemas novos que os sucedem. A respeito das técnicas de reengenharia, julgue os itens abaixo.

1. A realização de engenharia reversa está condicionada à existência de documentação adequada para o sistema de informação.

2. A reestruturação de código permite, de modo geral, elevar os níveis de compreensão acerca de um sistema sem documentaçãc a ponto de permitir a elaboração de documentação adequada.

3. Entre os preceitos da engenharia direta de um sistema de informação, estão a concepção de modificações incrementais, partir do código e da documentação existentes.

4. Independentemente do sistema em consideração, o custo de um processo de reengenharia é muito menor que o custo de uma nova concepção do sistema desde o princípio.

5. A manutenção de código-fonte é sempre uma forma, ainda que rudimentar, de reengenharia, visto que novas características são normalmente agregadas a partir desse processo.

06. O desenvolvimento de novos sistemas a partir da reengenharia de sistemas de informação e com a utilização de técnicas e de método, adequados pode significar uma grande economia no processo de transferência do conhecimento acumulado nos sistemas legados para os sistemas novos que os sucedem. A respeito das técnicas de reengenharia, julgue os itens abaixo.

1. ~~A realização de engenharia reversa está condicionada à existência de documentação adequada para o sistema de informação.~~

2. ~~A reestruturação de código permite, de modo geral, elevar os níveis de compreensão acerca de um sistema sem documentaçãc a ponto de permitir a elaboração de documentação adequada.~~

3. Entre os preceitos da engenharia direta de um sistema de informação, estão a concepção de modificações incrementais, partir do código e da documentação existentes.

4. ~~Independentemente do sistema em consideração, o custo de um processo de reengenharia é muito menor que o custo de uma nova concepção do sistema desde o princípio.~~

5. ~~A manutenção de código-fonte é sempre uma forma, ainda que rudimentar, de reengenharia, visto que novas características são normalmente agregadas a partir desse processo.~~

07. A substituição de sistemas legados por sistemas mais atualizados muitas vezes extrapola o escopo de utilização de técnicas de adaptação convencionais, sendo necessária a utilização de técnicas de reengenharia que abordem a reconcepção do sistema como um todo. Com referência às técnicas de reengenharia, assinale a opção correta.

A. Técnicas de reestruturação de código-fonte podem ser aplicadas com o auxílio de ferramentas especializadas, constituindo uma das principais formas de engenharia reversa.

B. No ciclo de vida de um sistema, a reengenharia é um processo estritamente relacionado às fases de análise e modelagem do sistema.

C. O processo de reengenharia de um software normalmente tem por objetivo obter um novo software, com maior qualidade que o seu precedente.

D. A opção pela reengenharia de um software deve ser adotada somente quando os custos desse processo forem menores que os custos de manutenção corretiva e adaptativa do software.

E. A engenharia reversa consiste em extrair diferentes níveis de abstração a partir de um código-fonte preexistente, com o objetivo de fundamentar novas implementações mais eficientes para o mesmo código-fonte.

07. A substituição de sistemas legados por sistemas mais atualizados muitas vezes extrapola o escopo de utilização de técnicas de adaptação convencionais, sendo necessária a utilização de técnicas de reengenharia que abordem a reconcepção do sistema como um todo. Com referência às técnicas de reengenharia, assinale a opção correta.

A. Técnicas de reestruturação de código-fonte podem ser aplicadas com o auxílio de ferramentas especializadas, constituindo uma das principais formas de engenharia reversa.

B. No ciclo de vida de um sistema, a reengenharia é um processo estritamente relacionado às fases de análise e modelagem do sistema.

C. O processo de reengenharia de um software normalmente tem por objetivo obter um novo software, com maior qualidade que o seu precedente.

D. A opção pela reengenharia de um software deve ser adotada somente quando os custos desse processo forem menores que os custos de manutenção corretiva e adaptativa do software.

E. A engenharia reversa consiste em extrair diferentes níveis de abstração a partir de um código-fonte preexistente, com o objetivo de fundamentar novas implementações mais eficientes para o mesmo código-fonte.

08. No tocante à reengenharia de sistemas, julgue os itens a seguir.

I - A reengenharia de sistemas está ligada à reengenharia de processos de negócios, pois os softwares são, com frequência, a representação informática de regras de negócios.

II – A manutenção do software compreende ações corretivas de adaptação, de aperfeiçoamento e de reengenharia.

III - A reengenharia de sistemas de informação é uma atividade que absorve recursos de tecnologias da informação durante longos períodos.

IV - A manutenção de uma documentação atualizada do software tem pouca influência sobre o processo de reengenharia, principalmente se o software for orientado a objetos.

V - A engenharia reversa de um software é frequentemente necessária nos processos de downsizing.

Estão certos apenas os itens

A. I e V
B. IV e V
C. I, II e III
D. I, II e IV
E. II, III e V

08. No tocante à reengenharia de sistemas, julgue os itens a seguir.

I - A reengenharia de sistemas está ligada à reengenharia de processos de negócios, pois os softwares são, com frequência, a representação informática de regras de negócios.

II – A manutenção do software compreende ações corretivas de adaptação, de aperfeiçoamento e de reengenharia.

III - A reengenharia de sistemas de informação é uma atividade que absorve recursos de tecnologias da informação durante longos períodos.

IV - A manutenção de uma documentação atualizada do software tem pouca influência sobre o processo de reengenharia, principalmente se o software for orientado a objetos.

V - A engenharia reversa de um software é frequentemente necessária nos processos de downsizing.

Estão certos apenas os itens

A. I e V
B. IV e V
C. I, II e III
D. I, II e IV
E. II, III e V

09.

I. A reengenharia reversa extrai informações de projeto de código-fonte quando nenhuma outra documentação encontra-se à disposição.

Porque

II. A reengenharia pega as informações extraídas dos programas existentes e reestrutura os mesmos para conseguir uma qualidade mais alta e, por conseguinte, melhorar manutenibilidade para o futuro.

A. Se I e II forem verdadeiras e II for uma justificativa correta de I.
B. Se I e II forem verdadeiras e II for uma justificativa INCORRETA de I.
C. Se I for verdadeira e II for falsa.
D. Se I for falsa e II for verdadeira.
E. Se I e II forem falsas.

09.

I. A reengenharia reversa extrai informações de projeto de código-fonte quando nenhuma outra documentação encontra-se à disposição.

Porque

II. A reengenharia pega as informações extraídas dos programas existentes e reestrutura os mesmos para conseguir uma qualidade mais alta e, por conseguinte, melhorar manutenibilidade para o futuro.

A. Se I e II forem verdadeiras e II for uma justificativa correta de I.
B. Se I e II forem verdadeiras e II for uma justificativa INCORRETA de I.
C. Se I for verdadeira e II for falsa.
D. Se I for falsa e II for verdadeira.
E. Se I e II forem falsas.

10. O segmento da engenharia de configuração que trata do teste e da manutenção de software é uma das áreas que mais demandam recursos das organizações. A respeito desse assunto, julgue os itens que se seguem.

1. A gerência de configuração é responsável por identificar, organizar e controlar as modificações a serem feitas em um software desenvolvido por uma equipe de desenvolvimento, com o objetivo de maximizar a produtividade e minimizar os erros.
2. Os métodos de teste de software denominados white box focalizam os requisitos funcionais do software; esse tipo de teste procura analisar todas as condições relacionadas aos requisitos funcionais de um programa.
3. O método de teste de software denominado black box utiliza a estrutura dos programas para realizar o teste, levando em conta as decisões lógicas, os loops e seus , limites e o teste das estruturas de dados para assegurar sua validade.
4. A reengenharia de software recupera a informação do projeto de software existente e a usa para alterar ou reconstruir o sistema existente, no esforço de melhorar a qualidade.
5. A engenharia reversa para software é o processo de analisar um programa para criar a sua representação em , um nível maior de abstração que o código-fonte.

10. O segmento da engenharia de configuração que trata do teste e da manutenção de software é uma das áreas que mais demandam recursos das organizações. A respeito desse assunto, julgue os itens que se seguem.

1. **A gerência de configuração é responsável por identificar, organizar e controlar as modificações a serem feitas em um software desenvolvido por uma equipe de desenvolvimento, com o objetivo de maximizar a produtividade e minimizar os erros.**
2. ~~Os métodos de teste de software denominados white box focalizam os requisitos funcionais do software; esse tipo de teste procura analisar todas as condições relacionadas aos requisitos funcionais de um programa.~~
3. ~~O método de teste de software denominado black box utiliza a estrutura dos programas para realizar o teste, levando em conta as decisões lógicas, os loops e seus , limites e o teste das estruturas de dados para assegurar sua validade.~~
4. **A reengenharia de software recupera a informação do projeto de software existente e a usa para alterar ou reconstruir o sistema existente, no esforço de melhorar a qualidade.**
5. **A engenharia reversa para software é o processo de analisar um programa para criar a sua representação em , um nível maior de abstração que o código-fonte.**

GABARITO

1. C
2. B
3. C, E, E, C, E
4. E, C, C, E, C
5. E
6. E, E, C, E, E
7. C
8. C
9. D
10. C, E, E, C, C