


Engenharia de Software I

Rogério Eduardo Garcia
(rogerio.garcia@unesp.br)

Aula 07


*To reach a port, we must sail – sail,
not tie at anchor - sail, not drift.*
Franklin D. Roosevelt



Bacharelado em
Ciência da
Computação
2025

1


Cronograma



30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

2




unesp
BCC
2025

2

Engenharia de Software I – Aula 7

- SQA
- Teste de Software



30/05/2025


Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

3

3

O Caminho...

Sumário



01/04/22

Ciência da Computação - Engenharia de Sof

✓

CAPÍTULO 1 A natureza do software 1

1.1 A natureza do software 3

1.1.1 Definição de software 4

1.1.2 Campos de atuação de software 6

1.1.3 Software legado 7

1.2 A natureza mutante do software 9

1.2.1 WebApps 9

1.2.2 Aplicativos móveis 9

1.2.3 Computação em nuvem 10

1.2.4 Software para infra de produtos (de software) 11

1.3 Resumo 11

Problemas e pontos a ponderar 12

Leituras e fontes de informação complementares 12

✓

CAPÍTULO 2 Engenharia de software 14

2.1 Definição da disciplina 15

2.2 O processo de software 16

2.2.1 A metodologia do processo 17

2.2.2 Atividades de apoio 18

2.2.3 Adaptação de processos 18

2.3 A prática da engenharia de software 19

2.3.1 A avaliação da prática 19

2.3.2 Princípios gerais 21

2.4 Mitos do desenvolvimento de software 23

2.5 Como tudo começa 26

2.6 Resumo 27

Problemas e pontos a ponderar 27

Leituras e fontes de informação complementares 27

✓

PARTE I O processo de software 29

CAPÍTULO 3 Estrutura do processo de software 30

3.1 Um modelo de processo genérico 31

3.2 Definição de uma atividade metodológica 32

3.3 Identificação de um conjunto de tarefas 34

3.4 Fluxos de processo 34

3.5 Avaliação e aperfeiçoamento de processos 37

3.6 Resumo 38

Problemas e pontos a ponderar 38

Leituras e fontes de informação complementares 39

✓

CAPÍTULO 4 Modelos de processo 40

4.1 Modelos de processo prescritivos 41

4.1.1 O modelo cascata 41

4.1.2 Modelos de processo incremental 43

4.1.3 Modelos de processo evolucionário 44

4.1.4 Modelos concorrentes 49

4.1.5 Um comentário final sobre processos evolucionários 51

4.2 Modelos de processo especializado 52

4.2.1 Desenvolvimento baseado em componentes 52

4.2.2 O modelo de métodos formais 53

4.2.3 Desenvolvimento de software orientado a aspectos 54

4.3 O processo unificado 55

4.3.1 Um breve histórico 56

4.3.2 Fases do processo unificado 56

4.4 Modelos de processo pessoal e da equipe 58

4.4.1 Processo de Software Pessoal 59

4.4.2 Processo de Software de Equipe 60

4.5 Tipologia de processos 61

4.6 Produto e processo 62

4.7 Resumo 63

Problemas e pontos a ponderar 64

Leituras e fontes de informação complementares 65

✓

CAPÍTULO 5 Desenvolvimento ágil 66

5.1 O que é agilidade? 68

5.2 Agilidade e o custo das mudanças 68

5.3 O que é processo ágil? 69

5.3.1 Princípios da agilidade 70

5.3.2 A prática do desenvolvimento ágil 71

5.4 Extreme programming – XP (Programação Extrema) 72

5.4.1 O processo XP 72

5.4.2 Indústria XP 76

5.5 Outros modelos de processos Ágeis 77

5.5.1 Scrum 78

5.5.2 Método de Desenvolvimento de Sistemas Dinâmicos (DSDM) 79

5.5.3 Modelagem Ágil (MA) 80

5.5.4 Processo Unificado Ágil 82

5.7 Resumo 84

Problemas e pontos a ponderar 85

Leituras e fontes de informação complementares 85

✓

CAPÍTULO 6 Aspectos humanos da engenharia de software 87

6.1 Características de um engenheiro de software 88

6.2 A psicologia da engenharia de software 89

6.3 A equipe de software 90

6.4 Estruturas de equipe 92

6.5 Equipes ágeis 93

6.5.1 A equipe ágil genérica 93

6.5.2 A equipe XP 94

6.6 O impacto da mídia social 95

⊘

4

O Caminho...

Sumário

xlvi

6.7 Engenharia de software usando a navegação 97

6.8 Ferramentas de colaboração 98

6.9 Equipos globais 99

6.10 Resumo 100

Problemas e pontos a ponderar 101

Lecturas e fontes de informação complementares 102

PARTE II Modelagem 103

CAPÍTULO 7 Princípios que orientam a prática 104

7.1 Conhecimento da engenharia de software 105

7.2 Princípios fundamentais 106

7.2.1 Princípios que orientam a prática 106

7.2.2 Princípios que orientam a prática 107

7.3 Princípios das atividades metodológicas 109

7.3.1 Princípios de comunicação 110

7.3.2 Princípios de planeamento 112

7.3.3 Princípios de modelagem 114

7.3.4 Princípios de construção 121

7.3.5 Princípios de disponibilização 124

7.4 Formas de trabalhar 125

7.5 Resumo 127

Problemas e pontos a ponderar 128

Lecturas e fontes de informação complementares 129

CAPÍTULO 8 Entendendo os requisitos 131

8.1 Engenharia de requisitos 132

8.2 Estabelecimento da base de trabalho 138

8.2.1 Identificação de requisitos 139

8.2.2 Reconhecimento de diversos pontos de vista 139

8.2.3 Trabalho em busca da compreensão 140

8.2.4 Questões iniciais 140

8.2.5 Requisitos das funcionalidades 141

8.2.6 Retornabilidade 142

8.3 Levantamento de requisitos 142

8.3.1 Colecção colaborativa de requisitos 143

8.3.2 Aplicação de questionário para QFD (Quality Function Deployment) 146

8.3.3 Criação de uso 146

8.3.4 Análise de levantamento de requisitos 147

8.3.5 Levantamento de requisitos a priori 148

8.3.6 Métodos orientados a processos 148

8.4 Desenvolvimento de casos de uso 149

8.4.1 Estrutura de modelos de análise 154

8.4.2 Padrões de análise 157

8.4.3 Engenharia de requisitos a priori 158

8.4.4 Requisitos de sistema activacionais 158

8.6 Negociação de requisitos 159

8.7 Monitoramento de requisitos 160

8.8 Validação dos requisitos 161

xlvi

Sumário

6.9 Fonte de informação 162

6.10 Resumo 162

Problemas e pontos a ponderar 163

Lecturas e fontes de informação complementares 164

CAPÍTULO 9 Modelagem de requisitos: métodos baseados em cenários 166

9.1 Análise de requisitos 167

9.1.1 Filosofia e objetivos gerais 168

9.1.2 Regras práticas para a análise 168

9.1.3 Análise de domínio 169

9.1.4 Algoritmos de modelagem de requisitos 171

9.2 Modelagem baseada em cenários 173

9.2.1 Criação de um caso de uso preliminar 173

9.2.2 Refinamento de um caso de uso preliminar 176

9.2.3 Criação de um caso de uso formal 177

9.3 Modelos UML que complementam o caso de uso 179

9.3.1 Desenvolvimento de um diagrama de atividades 179

9.3.2 Diagramas de rede 180

9.4 Resumo 182

Problemas e pontos a ponderar 182

Lecturas e fontes de informação complementares 183

CAPÍTULO 10 Modelagem de requisitos: métodos baseados em classes 184

10.1 Identificação das classes de análise 185

10.2 Especificação de atributos 189

10.3 Definição das operações 189

10.4 Modelagem classe-responsabilidade-colaborador 192

10.5 Associações e dependências 198

10.6 Factores de análise 199

10.7 Resumo 200

Problemas e pontos a ponderar 201

Lecturas e fontes de informação complementares 201

CAPÍTULO 11 Modelagem de requisitos: comportamento, padrões e WebApps/aplicativos móveis 202

11.1 Criação de um modelo comportamental 203

11.2 Identificação de eventos com o caso de uso 203

11.3 Representações de estados 204

11.4 Padrões para a modelagem de requisitos 207

11.4.1 Decisões de padrões de estados 208

11.4.2 Exemplos de padrões de requisitos: Algoritmo Sensor 209

11.5 Modelagem de requisitos para WebApps e aplicativos móveis 213

11.5.1 Que nível de análise é suficiente? 214

11.5.2 Requisitos de requisitos 214

11.5.3 Solução de modelagem de requisitos 215

11.5.4 Modelagem de requisitos 216

11.5.5 Modelo de interação para WebApps e aplicativos móveis 217

11.5.6 Modelo funcional 218

11.5.7 Modelo de configuração para WebApps 219

11.5.8 Modelo de navegação 220

01/04/22

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

5

O Caminho...

Sumário

xv

11.6 Resumo 221

Problemas e pontos a ponderar 222

Lecturas e fontes de informação complementares 222

CAPÍTULO 12 Conceitos de projeto 224

12.1 Projeto no contexto da engenharia de software 225

12.2 O processo de projeto 228

12.2.1 Diretrizes e atributos da qualidade de software 228

12.2.2 A avaliação de um projeto de software 229

12.3 Conceitos de projeto 231

12.3.1 Abstração 231

12.3.2 Arquitetura 232

12.3.3 Partição 233

12.3.4 Separação por interesses (por afinidade) 234

12.3.5 Modularidade 234

12.3.6 Encapsulamento de informações 235

12.3.7 Independência funcional 236

12.3.8 Refinamento 237

12.3.9 Aspectos 237

12.3.10 Referências 238

12.3.11 Conceitos de projeto orientados a objetos 238

12.3.12 Classes de projeto 239

12.3.13 Invenção de dependência 241

12.3.14 Projeto para teste 242

12.4 O modelo de projeto 243

12.4.1 Elementos de projeto de dados 244

12.4.2 Elementos de projeto de arquitetura 244

12.4.3 Elementos de projeto de interface 245

12.4.4 Elementos de projeto de componentes 247

12.4.5 Elementos de projeto de implementação 247

12.5 Resumo 249

Problemas e pontos a ponderar 250

Lecturas e fontes de informação complementares 250

CAPÍTULO 13 Projeto de arquitetura 252

13.1 Arquitetura de software 253

13.1.1 O que é arquitetura? 253

13.1.2 Por que a arquitetura é importante? 254

13.1.3 Descrição de arquitetura 255

13.1.4 Decisões de arquitetura 256

13.2 Categorias de arquitetura 257

13.3 Estilos de arquitetura 258

13.3.1 Uma breve taxonomia dos estilos de arquitetura 258

13.3.2 Padrões de arquitetura 260

13.3.3 Organização e refinamento 263

13.4 Considerações sobre a arquitetura 264

13.5 Decisões sobre a arquitetura 266

13.6 Projeto de arquitetura 267

13.6.1 Representação do sistema no contexto 267

13.6.2 Definição de requisitos 269

13.6.3 Refinamento da arquitetura em componentes 270

13.6.4 Descrição das instâncias do sistema 272

xv

Sumário

13.6.5 Projeto de arquitetura para aplicações web (WebApps) 273

13.6.6 Projeto de arquitetura para aplicativos móveis 274

13.7 Avaliação das alternativas de projeto de arquitetura 274

13.7.1 Linguagens de descrição de arquitetura 276

13.7.2 Revisões de arquitetura 277

13.8 Lições aprendidas 278

13.9 Revisão de arquitetura baseada em padrões 278

13.10 Verificação de conformidade da arquitetura 279

13.11 Agilidade e arquitetura 280

13.12 Resumo 282

Problemas e pontos a ponderar 282

Lecturas e fontes de informação complementares 283

CAPÍTULO 14 Projeto de componentes 285

14.1 O que é componente? 286

14.1.1 Unidades elementares e abstratas 286

14.1.2 A vista tradicional 288

14.1.3 Unidades relacionadas a processos 291

14.2 Projeto de componentes baseados em classes 291

14.2.1 Princípios básicos de projeto 292

14.2.2 Diretrizes para o projeto de componentes 295

14.2.3 Criação 296

14.2.4 Acoplamento 298

14.3 Construção de projetos de componentes 299

14.3.1 Projeto de componentes para WebApps 305

14.3.2 Projeto de componentes para componentes 305

14.3.3 Projeto funcional para componentes 306

14.3.4 Projeto de componentes para aplicativos móveis 306

14.3.5 Projeto de componentes tradicionais 307

14.3.6 Desenvolvimento baseado em componentes 308

14.3.7 Engenharia de domínio 308

14.3.8 Classificação, seleção e composição de componentes 309

14.3.9 Divergência arquitetural 311

14.3.10 Análise e projeto para manutenção 312

14.3.11 Classificação e reorganização de componentes 312

14.3.12 Resumo 313

Problemas e pontos a ponderar 315

Lecturas e fontes de informação complementares 316

CAPÍTULO 15 Projeto de interfaces de usuário 317

15.1 Análise de usuário 318

15.1.1 Definir o usuário no contexto 318

15.1.2 Definir o contexto de uso do usuário 319

15.1.3 Definir a interface consistente 321

15.2 Análise e projeto de interfaces 322

15.2.1 Modelos de análise e projeto de interfaces 322

15.2.2 O processo 323

15.2.3 Análise de usuário 325

15.2.4 Análise e modelagem de interface 326

15.2.5 Análise do contexto de uso 327


15.2.6 Análise do ambiente de trabalho 327

01/04/22

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

6

O Caminho...



Símbolo

xxviii

Símbolo

xxviii

15.4 Etapas no projeto de interfaces 332

15.4.1 Avaliação das etapas para projeto de interfaces 332

15.4.2 Padrões de projeto de interfaces do usuário 334

15.4.3 Questões de projeto 335

15.5 Projeto de interfaces para WebApps e aplicativos móveis 337

15.5.1 Diretrizes e estruturas para projeto de interfaces 337

15.5.2 Fluxo de trabalho de projeto de interfaces para WebApps e aplicativos móveis 341

15.6 Avaliação do projeto 342

15.7 Resumo 344

Problemas e pontos a ponderar 345

Leituras e fontes de informação complementares 346

CAPÍTULO 16 Projeto baseado em padrões 347

16.1 Padrões de projeto 348

16.1.1 Tipos de padrões 349

16.1.2 Hierarquia 351

16.1.3 Descrição de padrões 352

16.1.4 Usagem e repositório de padrões 353

16.2 Projeto de software baseado em padrões 354

16.2.1 Contexto do projeto baseado em padrões 354

16.2.2 Pensar em termos de padrões 354

16.2.3 Síntese de projetos 356

16.2.4 Contribuição de uma tabela para organização de padrões 358

16.2.5 Erros comuns de projeto 359

16.3 Padrões de arquitetura 359

16.4 Padrões de projeto de componentes 360

16.5 Padrões de projeto para interfaces do usuário 362

16.6 Padrões de projeto para WebApps 364

16.6.1 Tipos de projetos 365

16.6.2 Generalização do projeto 365

16.6.3 Padrões para aplicativos móveis 366

16.6.4 Resumo 367

Problemas e pontos a ponderar 368

Leituras e fontes de informação complementares 369

CAPÍTULO 17 Projeto de WebApps 371

17.1 Qualidade do projeto em WebApps 372

17.2 Objetivos do projeto 374

17.3 Uma perspectiva do projeto para WebApps 375

17.4 Projeto de interfaces para WebApps 376

17.5 Projeto estático 377

17.5.1 Questões de design 378

17.5.2 Questões de design gráfico 378

17.6 Projeto de conteúdo 379

17.6.1 Objetos de conteúdo 379

17.6.2 Questões de projeto de conteúdo 380

17.7 Projeto de arquitetura 381

17.7.1 Arquitetura de conteúdo 381

17.7.2 Arquitetura de uma WebApp 384

17.8 Projeto de navegação 385

17.8.1 Semântica de navegação 385

17.8.2 Síntese de navegação 387

17.9 Projeto em nível de componentes 387

17.10 Resumo 388

Problemas e pontos a ponderar 389

Leituras e fontes de informação complementares 389

CAPÍTULO 18 Projeto de aplicativos móveis 391

18.1 Os desafios 392

18.1.1 Considerações sobre o desenvolvimento 392

18.1.2 Considerações técnicas 393

18.2 Desenvolvimento de aplicativos móveis 395

18.2.1 Qualidade de aplicativos móveis 397

18.2.2 Projeto de interfaces de usuário 398

18.2.3 Aplicativos sensíveis ao contexto 398

18.2.4 Locais geográficos 400

18.3 Projeto de aplicativos móveis – boas práticas 401

18.4 Ambientes de mobilidade 403

18.5 A nuvem 405

18.6 A aplicabilidade da engenharia de software convencional 407

18.7 Resumo 408

Problemas e pontos a ponderar 409

Leituras e fontes de informação complementares 410

PARTE III Gestão da qualidade 411

CAPÍTULO 19 Conceitos de qualidade 412

19.1 O que é qualidade? 413

19.2 Qualidade de software 414

19.2.1 Dimensões de qualidade de Garvin 415

19.2.2 Fatores de qualidade de NCSC 416

19.2.3 Fatores de qualidade ISO 9126 416

19.2.4 Fatores de qualidade de projeto 418

19.2.5 A transição para uma visão quantitativa 420

19.3 O dilema da qualidade do software 420

19.3.1 Software "born to succeed" 421

19.3.2 Custo da qualidade 422

19.3.3 Riscos 424

19.3.4 Responsabilidade e responsabilidade civil 425

19.3.5 Qualidade e segurança 425

19.3.6 O impacto das ações administrativas 426

19.4 Alcançando a qualidade de software 427

19.4.1 Métodos de engenharia de software 427

19.4.2 Técnicas de gerenciamento de software 427

19.4.3 Controle de qualidade 428

19.4.4 Capacidade de qualidade 428

19.5 Resumo 428

Problemas e pontos a ponderar 429


Leituras e fontes de informação complementares 430

01/04/22

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

7

Contextualizando... ISO 12207: Estrutura



Processos Fundamentais

Aquisição

Fornecimento

Desenvolvimento

Operação

Manutenção

Processos de Apoio

Documentação

Gerenciamento de Configuração

Garantia de Qualidade

Verificação

Validação

Revisão Conjunta

Auditoria

Resolução de Problemas

Processos Organizacionais

Gerência

Melhoria

Infra-estrutura

Treinamento

Adaptação

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

8

8

Prof. Dr. Rogério E. Garcia

4

Qualidade de Software



Conformidade com **requisitos funcionais** e **de desempenho** explicitamente declarados, **padrões de desenvolvimento** claramente documentados e **características implícitas** que são esperadas em todo software desenvolvido profissionalmente.

(Pressman, 2001)

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

9

9

Qualidade de Software



A definição de qualidade enfatiza três aspectos importantes:

Requisitos de software

Padrões de desenvolvimento

Requisitos implícitos

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

10

10

Qualidade de Software



Os **requisitos de software** são a base a partir da qual a qualidade é medida.

A falta de conformidade aos requisitos significa falta de qualidade.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

11

11

Qualidade de Software



Os **padrões de desenvolvimento** definem um conjunto de **critérios** que orientam a maneira segundo a qual o software passa pelo trabalho de engenharia.

Se os critérios não forem seguidos, o resultado quase que seguramente será a falta de qualidade.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

12

12

Qualidade de Software



Existe um conjunto de **requisitos implícitos** que frequentemente não são mencionados na especificação.

- Facilidade de uso.
- Desejo de uma boa manutenibilidade.

Se o software atende aos requisitos explícitos, mas falha nos requisitos implícitos, a qualidade é suspeita.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

13

13

Qualidade de Software



Existe, ainda, uma visão de qualidade de software do ponto de vista **gerencial**.

O software é considerado de qualidade desde que possa ser desenvolvido dentro do **prazo** e do **orçamento** especificados.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

14

14

Visões de Qualidade de Software

The diagram illustrates three stakeholder perspectives on software quality, each with a corresponding icon and a list of concerns:

- Usuário** (User): Represented by an icon of a person holding a green plant. Concerns include: Facilidade de uso, desempenho, confiabilidade dos resultados, preços do software, etc.
- Desenvolvedor** (Developer): Represented by an icon of a person at a laptop. Concerns include: Taxa de defeitos, facilidade de manutenção e conformidade em relação aos requisitos dos usuários, etc.
- Organização** (Organization): Represented by an icon of colorful buildings. Concerns include: Cumprimento de prazo, boa previsão de custo, boa produtividade.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

15

15

Gerenciamento de Qualidade

The slide lists three key tasks for quality management:

- Definir procedimentos e padrões a serem utilizados durante o desenvolvimento de software.
- Verificar se os mesmos estão sendo seguidos por todos os engenheiros.
- Desenvolver uma **cultura de qualidade**.
 - Todos estão **comprometidos** a atingir um determinado **nível** de qualidade para o produto.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

16

16

Gerenciamento de Qualidade



Estruturado em duas atividades principais:

Planejamento da Qualidade

Seleção de **procedimentos** e **padrões** organizacionais que conduzam a um software de alta qualidade.

Deve começar em um estágio inicial do processo de desenvolvimento.

Plano de Qualidade

Estabelece as qualidades desejadas para o produto.

Define como tais qualidades devem ser avaliadas.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

17

17

Gerenciamento de Qualidade



Estruturado em duas atividades principais:

Controle de Qualidade

Consiste em supervisionar o processo de desenvolvimento a fim de assegurar que os procedimentos e padrões de qualidade sejam seguidos pela equipe.


Envolve uma série de **inspeções**, **simulações** e **testes**.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

18

18



Custo da Qualidade

- Inclui todos os custos embutidos na qualidade ou para executar atividades relacionadas a ela.

Custos de Prevenção:

- planejamento da qualidade
- revisões técnicas formais
- equipamento de teste
- treinamento

Custos de Avaliação:

- inspeções *no* e *entre* processos
- manutenção de equipamentos
- teste

Custos de Falhas:


- retrabalho
- correção
- análise da falha

Custos de Falhas Externas:

- resolução da queixa
- retorno e troca de produto
- suporte de ajuda ao usuário
- garantia de trabalho

30/05/2025
Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia
19

19



Garantia de Qualidade de Software

(Software Quality Assurance – SQA)

Padrão planejado e sistemático de **ações** necessárias para garantir a qualidade do software.


O objetivo é fornecer à **gerência** os dados necessários sobre a qualidade do produto.

Se problemas forem identificados...

O gerente é responsável por aplicar os mecanismos necessários para resolver os problemas de qualidade.

30/05/2025
Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia
20

20




Atividades de SQA

Ações para garantir a qualidade do software:

- Aplicação de **métodos técnicos**.
 - Ajudar o analista a conseguir uma especificação de qualidade.
 - Ajudar o projetista a desenvolver um projeto de qualidade.
- Realização de **revisões**.
 - Avaliar a qualidade da especificação, do projeto, do código, ...
- Atividades de **teste de software**.
 - Ajudar a garantir que a detecção de erros seja efetiva.

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 21

21



Atividades de SQA

Ações para garantir a qualidade do software:

- Aplicação de **padrões e procedimentos formais**.
 - Garantir que estes sejam seguidos durante o desenvolvimento.
- Processo de **controle de mudanças**.
 - Atividade associada ao gerenciamento de configuração de software.
 - Formalizar pedidos de mudança.
 - Avaliar a natureza da mudança e controlar seu impacto.

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 22

22

Atividades de SQA



Ações para garantir a qualidade do software:

Mecanismos de **medição**.

Apoio no acompanhamento da qualidade de software.

Avaliar o impacto de mudanças metodológicas e procedimentais.

Anotação e manutenção de registros.

Procedimentos para coleta e disseminação de informações de garantia de qualidade.

Registro histórico do projeto.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

23

23

Atividades de SQA



A realização das atividades de SQA pode ser associada a dois grupos:

Engenheiros de Software.

Aplicação de métodos e medidas técnicas sólidas.

Condução de revisões.

Realização de testes sistemáticos e planejados.

Grupo de SQA.

Planejamento, supervisão, registro, análise e relato da garantia da qualidade.

Ajudam a equipe de software a conseguir um produto final de alta qualidade.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

24

24

Atividades de SQA



Atividades do Grupo de SQA:

- Preparar um **plano de SQA** para o projeto.
- Avaliações a serem executadas.
- Auditorias e revisões a serem executadas.
- Padrões que são aplicáveis ao projeto.
- Procedimentos para acompanhamento e relatório de erros.
- Documentos a serem produzidos pelo grupo de SQA.
- Realimentação fornecida à equipe de projeto.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

25

25

Atividades de SQA



Atividades do Grupo de SQA:

- Participar no desenvolvimento da descrição do processo de software adotado no projeto.
- Revisar as atividades de ES para verificar a conformidade com o processo de software definido.
- Auditar os produtos de trabalho técnico a fim de verificar a conformidade com o que foi definido como parte do processo do software.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

26

26

Atividades de SQA



Atividades do Grupo de SQA:

Assegurar que os desvios ocorridos sejam documentados e manipulados de acordo com um procedimento previamente estabelecido.

Registrar as não-conformidades e comunicá-las à gerência.

Coordenar o controle e o gerenciamento das mudanças.

Ajudar a coletar e analisar as métricas utilizadas.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

27

27

Verificação e Validação



Dentre as atividades de SQA estão as atividades de **verificação** e **validação** de software.

Atividades de V&V.

O objetivo é minimizar a ocorrência de erros e riscos associados.


Detectar a **presença de erros** nos produtos de software.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

28

28



Verificação e Validação

Verificação


Garantir consistência, completitude e corretitude do produto em cada fase e entre fases consecutivas do ciclo de vida do software.

Verificar se os métodos e processos de desenvolvimento foram adequadamente aplicados.

Estamos construindo certo o produto?

30/05/2025Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia29

29



Verificação e Validação


Validação

Assegurar que o produto final corresponda aos requisitos do usuário.

Estamos construindo o produto certo?

30/05/2025Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia30

30




Verificação e Validação

V&V abrangem um amplo conjunto de atividades de SQA:

- Revisões técnicas formais.
- Auditoria de qualidade e configuração.
- Monitoramento de desempenho.
- Simulação.
- Estudo de viabilidade.
- Revisão da documentação.
- Revisão da base de dados.
- Testes.

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 31

31



Verificação e Validação

V&V envolvem atividades de...

Análise estática.

- Não requerem a execução propriamente dita do produto.
- Podem ser aplicadas em qualquer produto intermediário do processo de desenvolvimento.
 - Documento de requisitos, diagramas de projeto, código-fonte, planos de teste, ...
- As revisões são o exemplo mais clássico de análise estática.

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 32

32

Verificação e Validação



V&V envolvem atividades...

Análise dinâmica.

- Requerem a execução do produto.

 - Código.

 - Quaisquer outras **representações executáveis** do sistema.

 - Especificação executável.

As atividades de simulação e teste constituem uma análise dinâmica do produto.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

33

33

Revisões de Software



Meio efetivo para melhorar a **qualidade** de software.

“**Filtro**” para o processo de Engenharia de Software.

Podem ser aplicadas em **vários pontos** durante o desenvolvimento do software.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

34

34

Revisões de Software



Maneira de usar a **diversidade** de um **grupo de pessoas** para:

- Apontar melhorias necessárias ao produto.

- Confirmar as partes de um produto em que uma melhoria não é desejada ou não é necessária.

- Realizar um trabalho técnico de qualidade mais uniforme de forma a torná-lo mais administrável.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

35

35

Revisões de Software



Objetivo principal

- Encontrar erros durante o processo de desenvolvimento, de forma que eles não se transformem em defeitos depois da entrega do software.

- Descoberta **precoce** dos erros.

 - Melhoria da qualidade** já nas primeiras fases do processo de desenvolvimento.

 - Produtividade e custo.**

 - Erros são detectados quando sua correção é mais **barata**.

 - Aumento da produtividade e diminuição dos custos.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

36

36

Revisões de Software



Além disso...

É uma **oportunidade de treinamento**.

“Aprender por experiência”.

Participantes aprendem as razões e padrões em descobrir erros.

Participantes aprendem bons padrões de desenvolvimento de software.

Com o decorrer do tempo....

A revisão auxilia os participantes a desenvolver produtos mais fáceis de entender e de manter.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

37

37

Revisões de Software



Tipos de Revisão

Discussão informal de um problema técnico.

Apresentação do projeto de software para uma audiência de clientes, administradores e pessoal técnico.

Revisões Técnicas Formais, as quais incluem avaliações técnicas do software realizadas em pequenos grupos.

Inspeção, **Walkthrough** e **Peer-Review** são métodos utilizados durante as Revisões Técnicas Formais.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

38


38

Revisões Técnicas Formais

Uma RTF é conduzida em uma reunião de revisão.

Independentemente do formato de RTF, toda reunião de revisão deve seguir as seguintes recomendações:

- Envolver de 3 a 5 pessoas.
- Deve haver uma preparação para a reunião.
 - A preparação não deve exigir mais de 2 horas de trabalho de cada pessoa.
- A reunião deve durar menos de 2 horas.
- Deve-se focalizar uma parte específica do software.
 - Maior probabilidade de descobrir erros.




30/05/2025

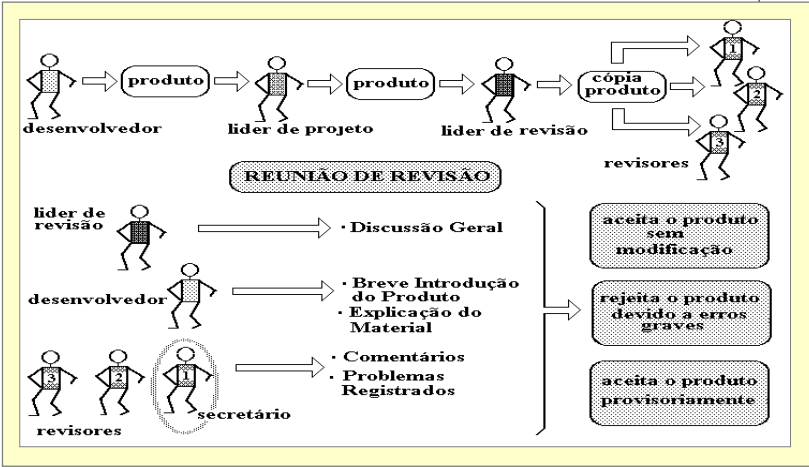
Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

39

39

Revisões Técnicas Formais






30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

40

40




Diretrizes para Revisão

- ✓ Revise o produto, não o produtor.
- ✓ Fixe e mantenha uma agenda.
- ✓ Limite o debate e a refutação.
- ✓ Relacione as áreas problemáticas.
- ✓ Faça anotações por escrito.
- ✓ Limite o número de participantes e insista em uma preparação antecipada.

30/05/2025Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia41

41



Diretrizes para Revisão

- ✓ Desenvolva uma lista de conferência (*checklist*) para cada produto que provavelmente será revisto.
- ✓ Atribua recursos e uma programação de tempo para as revisões.
- ✓ Realize um treinamento significativo para todos os revisores.
- ✓ Reveja suas antigas revisões.

30/05/2025Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia42

42




Revisão Técnica Formal

Métodos de RTF:

- Inspeção.*
- Walkthrough.*
- Peer-Review.*

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 43

43



Inspeção de Software

Método de **análise estática** para verificar a qualidade de um produto de software.

Detecção de Defeitos

Como detectar defeitos?

- ⇒ Lendo o documento.
- ⇒ Entendendo o que o documento descreve.
- ⇒ Checando as propriedades de qualidade requeridas.

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 44

44

Walkthrough



Procedimento similar ao procedimento para condução de uma inspeção.

A diferença fundamental está na maneira como a sessão de revisão é conduzida.

Em vez de simplesmente ler o programa ou checar os erros por meio de um *checklist*, os participantes **simulam** sua execução.

Papel adicional: Testador.

Elaborar um pequeno conjunto de casos de teste (em papel).

Monitorar e controlar os resultados obtidos.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

45

45

Peer-Review



Conduzida por **pares de programadores**.

Mesmo nível de conhecimento.

Aplicada ao **código**.

Reuniões com duração de 1 a 2 horas.

Somente um programa ou parte dele (rotinas) deve ser revisado.

Resultados são publicados em um relatório informal.

Não faz parte da documentação oficial do projeto.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

46

46

Teste de Software



Análise **dinâmica** do produto de software.

Processo de **executar** o software de modo controlado, observando seu comportamento em relação aos requisitos especificados.

Objetivo: revelar a presença de erros.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

47

47

SQA no Processo de Desenvolvimento de Software



Aspectos Positivos 😊

O software terá menos **defeitos** latentes.

Maior **confiabilidade** resultará em maior satisfação do cliente.

O **custo do ciclo de vida global** do software é reduzido.

Os **custos de manutenção** podem ser reduzidos.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

48

48

SQA no Processo de Desenvolvimento de Software



Aspectos Negativos ☹️

Difícil de ser instituída em **pequenas empresas**.

Representa uma **mudança cultural**.

Mudança nunca é fácil!!!

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

49

49

Teste de Software



Introdução

Teste de Software

Terminologia e Conceitos Básicos

Técnicas e Critérios de Teste

Automatização da Atividade de Teste

Estudos Empíricos

Perspectivas


Adaptado a partir do material preparado pelo Prof. Dr. Paulo C. Masiero – ICMC/USP

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

50

50



Introdução


Qualidade de Software

Conformidade com requisitos funcionais e de desempenho, padrões de desenvolvimento documentados e características implícitas esperadas de todo software profissionalmente desenvolvido.

- Corretitude
- Confiabilidade
- Testabilidade

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 51

51



Introdução

Garantia de Qualidade de Software


Conjunto de atividades técnicas aplicadas durante todo o processo de desenvolvimento

Objetivo

Garantir que tanto o processo de desenvolvimento quanto o produto de software atinjam os níveis de qualidade especificados

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 52

52



Introdução

Verificação: Assegurar consistência, completude e correitude do produto em cada fase e entre fases consecutivas do ciclo de vida do software

Estamos construindo corretamente o produto?


Validação: Assegurar que o produto final corresponda aos requisitos do usuário

Estamos construindo o produto certo?

Teste: Examina o comportamento do produto por meio de sua execução

30/05/2025Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia53

53



Terminologia

Defeito ➡ Erro ➡ Falha

Defeito: deficiência mecânica ou algorítmica que, se ativada, pode levar a uma falha

Erro: item de informação ou estado de execução inconsistente

Falha: evento notável em que o sistema viola suas especificações

30/05/2025Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia54

54

Defeitos no Processo de Desenvolvimento



A maior parte é de origem humana

São gerados na comunicação e na transformação de informações

Continuam presentes nos diversos produtos de software produzidos e liberados (10 defeitos a cada 1000 linhas de código)

A maioria encontra-se em partes do código raramente executadas

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

55

55

Defeitos no Processo de Desenvolvimento



Principal causa: tradução incorreta de informações

Quanto antes a presença do defeito for revelada, menor o custo de correção do defeito e maior a probabilidade de corrigi-lo corretamente


Solução: introduzir atividades de VV ao longo de todo o ciclo de desenvolvimento

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

56

56



Teste e Depuração

Teste

Processo de execução de um programa com o objetivo de revelar a presença de erros.


Contribuem para aumentar a confiança de que o software desempenha as funções especificadas.

Depuração

Consequência não previsível do teste. Após revelada a presença do erro, este deve ser encontrado e corrigido.

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 57

57



Teste de Software

Fundamental em todos os ramos de engenharia
Software: produto da Engenharia de Software

Atividade essencial para ascensão ao nível 3 do Modelo CMM/SEI

Atividade relevante para funcionalidade e sua avaliação


Qualidade de Software: ISO 9126 – substituída pela ISO 25010 e 25002:2024

Avaliação da qualidade: 14598-5

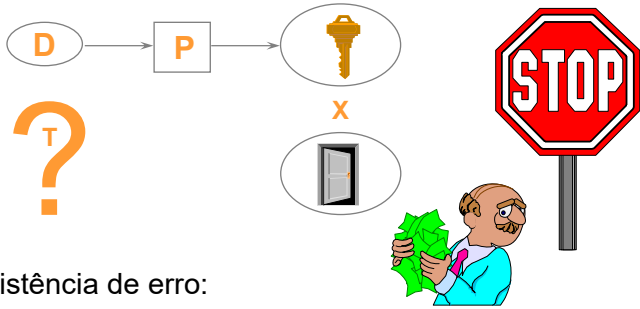
30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 58

58

Teste de Software



Objetivo: revelar a presença de erros




Inexistência de erro:
Software é de alta qualidade?
Conjunto de casos de teste T é de baixa qualidade?

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 59

59

Teste de Software



Defeitos e erros não revelados

- Falhas se manifestam durante a utilização pelos usuários
- Erros devem ser corrigidos durante a manutenção

Alto custo

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 60

60

Teste de Software



Falhas graves

- Qualidade e confiabilidade suspeitas
- Modificação do projeto
- Novos testes

Erros de fácil correção

- Funções aparentemente funcionam bem
- Qualidade e confiabilidade aceitáveis
- Testes inadequados para revelar a presença de erros graves
- Novos testes

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

61

61

Teste de Software



Limitações

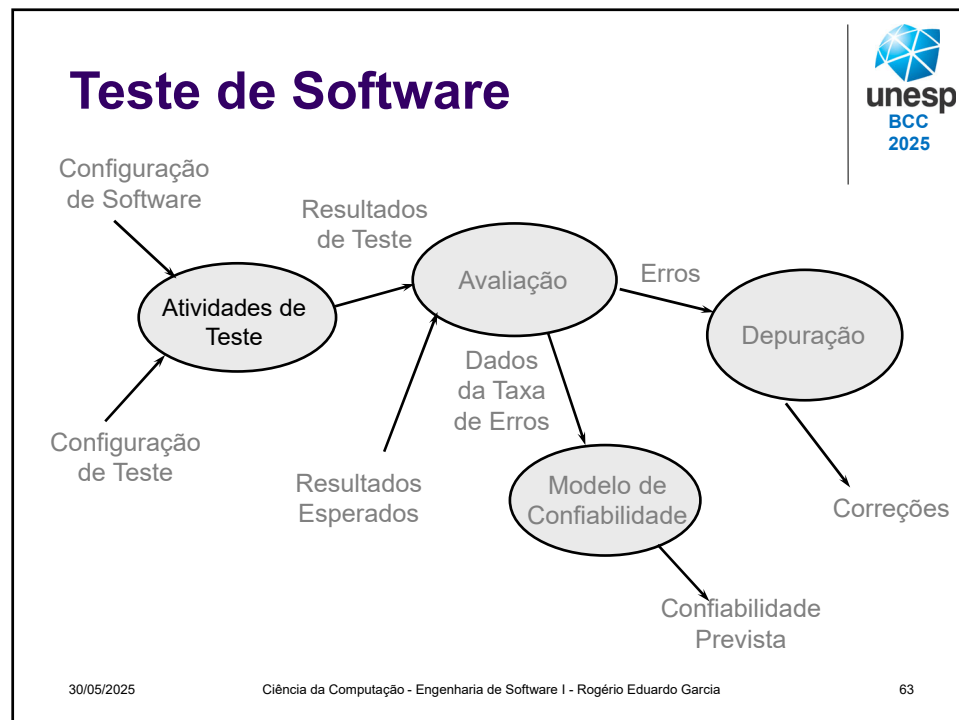
- Não existe um algoritmo de teste de propósito geral para provar a corretude de um programa
- Em geral, é indecidível se dois caminhos de um mesmo programa ou de diferentes programas computam a mesma função
- É indecidível se existe um dado de entrada que leve à execução de um dado caminho de um programa; isto é, é indecidível se um dado caminho é executável ou não

30/05/2025

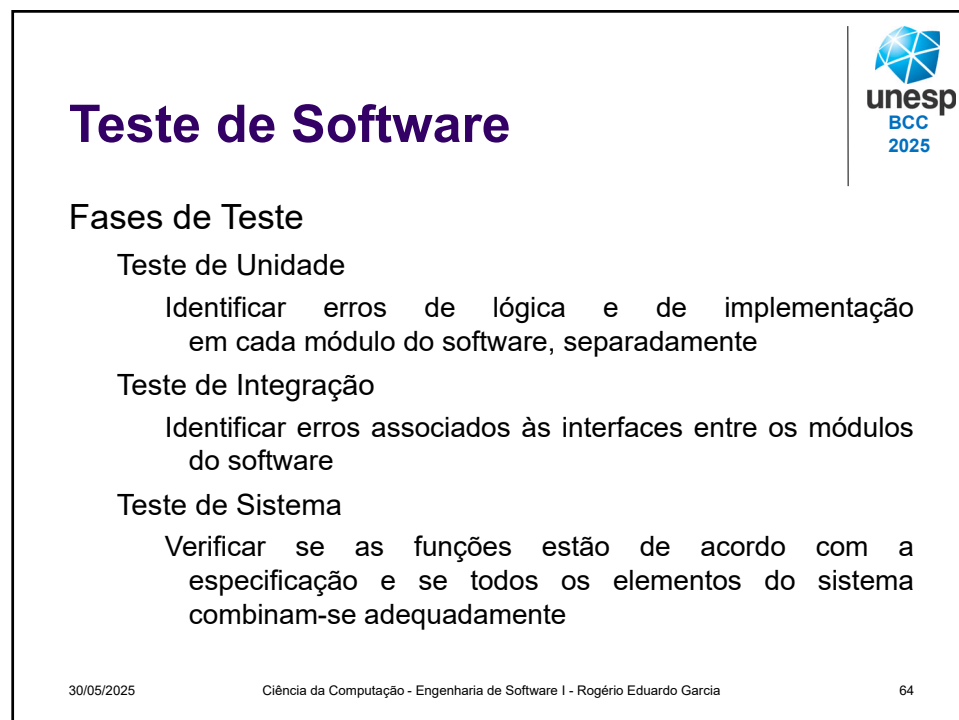
Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

62

62



63



64

Teste de Software



Etapas do Teste

Planejamento

Projeto de casos de teste

Execução do programa com os casos de teste

Análise de resultados

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

65

65

Teste de Software



Caso de teste

Especificação de uma entrada para o programa e a correspondente saída esperada

Entrada: conjunto de dados necessários para uma execução do programa

Saída esperada: resultado de uma execução do programa

Oráculo

Um bom caso de teste tem alta probabilidade de revelar um erro ainda não descoberto

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

66

66

Teste de Software



Projeto de casos de teste

- O projeto de casos de teste pode ser tão difícil quanto o projeto do próprio produto a ser testado
- Poucos programadores/analistas gostam de teste e, menos ainda, do projeto de casos de teste
- O projeto de casos de teste é um dos melhores mecanismos para a prevenção de defeitos
- O projeto de casos de teste é tão eficaz em identificar erros quanto a execução dos casos de teste projetados

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

67

67

Teste de Software



Maneira sistemática e planejada para conduzir os testes

Técnicas e Critérios de Teste

Conjunto de Casos de Teste T

Características desejáveis

Deve ser finito

Custo de aplicação deve ser razoável

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

68

68

Técnicas e Critérios de Teste



Critério de Teste C

Objetivo

Obter, de maneira sistemática, um conjunto T de casos de teste que seja efetivo quanto à meta principal de teste (revelar a presença de erros no programa)

Propriedades

- i) incluir todos os desvios de fluxo de execução
- ii) incluir pelo menos um uso de todo resultado computacional
- iii) T mínimo e finito

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

69

69

Técnicas e Critérios de Teste



Critério de Seleção de Casos de Teste

Procedimento para escolher casos de teste para o teste de P

Critério de Adequação

Predicado para avaliar T no teste de P

T é C -adequado \Leftrightarrow todo elemento requerido por C é exercitado por pelo menos por um $t, t \in T$

30/05/2025


Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

70

70

Técnicas e Critérios de Teste

- Técnica Funcional
 - Requisitos funcionais do software
 - Critério Particionamento em Classes de Equivalência
- Técnica Estrutural
 - Estrutura interna do programa
 - Critérios Baseados em Fluxo de Dados
- Técnica Baseada em Erros
 - Erros mais frequentes cometidos durante o processo de desenvolvimento de software
 - Critério Análise de Mutantes



unesp
BCC
2025

30/05/2025


Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

71

71

Técnica Funcional (Caixa Preta)

- Baseia-se na especificação do software para derivar os requisitos de teste
- Aborda o software de um ponto de vista macroscópico
- Envolve dois passos principais:
 - Identificar as funções que o software deve realizar (especificação dos requisitos)
 - Criar casos de teste capazes de checar se essas funções estão sendo executadas corretamente



unesp
BCC
2025

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

72

72

Técnica Funcional



Problema

Dificuldade em quantificar a atividade de teste: não se pode garantir que partes essenciais ou críticas do software foram executadas

Dificuldade de automatização

Critérios da Técnica Funcional

Particionamento em Classes de Equivalência

Análise do Valor Limite

Grafo de Causa-Efeito

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

73

73

Técnica Funcional: Exemplo



Particionamento em Classes de Equivalência

Divide o domínio de entrada do programa em classes de dados (classes de equivalências)

Os dados de teste são derivados a partir das classes de equivalência

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

74

74

Técnica Funcional: Exemplo



Passos

Identificar classes de equivalência

Condições de entrada

Classes válidas e inválidas

Definir os casos de teste

Enumeram-se as classes de equivalência

Casos de teste para as classes válidas

Casos de teste para as classes inválidas

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

75

75

Técnica Funcional: Exemplo



Especificação do programa *Identifier*

O programa deve determinar se um identificador é válido ou não em Silly Pascal (uma variante do Pascal). Um identificador válido deve começar com uma letra e conter apenas letras ou dígitos. Além disso, deve ter no mínimo um caractere e no máximo seis caracteres de comprimento.

- Exemplo

abc12 (válido);
cont*1 (inválido);

1soma (inválido);
a123456 (inválido)

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

76


76

Técnica Funcional: Exemplo

Classes de equivalência

Condições de Entrada	Classes Válidas	Classes Inválidas
Tamanho t do identificador	$1 \leq t \leq 6$ (1)	$t > 6$ (2)
Primeiro caractere c é uma letra	Sim (3)	Não (4)
Só contém caracteres válidos	Sim (5)	Não (6)

- Exemplo de Conjunto de Casos de Teste
 - $T_0 = \{(a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido})\}$
(1, 3, 5) (4) (6) (2)



30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

77

77


Técnica Funcional: Análise de valor-limite

Observações da prática profissional mostram que grande parte dos erros ocorre nas fronteiras do domínio de entrada

Completa a técnica de classes de equivalência

Casos de teste são selecionados nas bordas da classe.

Também deriva casos de testes para a saída.




30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

78

78

Técnica Funcional: Análise de valor-limite



Se os limites da condição de entrada forem a e b, projetar c.t. para os valores imediatamente acima e abaixo de a e b.

Se uma condição de entrada especifica vários valores , projetar casos de teste para os valores imediatamente acima e abaixo do valor mínimo e do valor máximo.

Aplicar as mesmas diretrizes para os valores de saída.

Se as estruturas de dados internas do programa têm limites especificados, projeto um caso de teste para exercitar a estrutura de dados no seu limite.


30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

79

79

Técnica Funcional: Exemplo



Classes de equivalência

Condições de Entrada	Classes Válidas	Classes Inválidas
Tamanho t do identificador	$t=1, 6$ (1)	$t=0, 7$ (2)
Primeiro caractere c é uma letra	--	--
Só contém caracteres válidos	--	--

- Exemplo de Conjunto de Casos de Teste
 - $T_0 = \{ (a, \text{Válido}), (Abcdef, \text{Válido}), (, \text{Inválido}), (abcdefg, \text{Inválido}) \}$

(1) (1) (2) (2)

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

80

80

Técnica Estrutural (Caixa Branca)



Baseada no conhecimento da estrutura interna (implementação) do programa

Teste dos detalhes procedimentais

A maioria dos critérios dessa técnica utiliza uma representação de programa conhecida como grafo de programa ou grafo de fluxo de controle

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

81

81

Técnica Estrutural



Grafo de Programa

Nós: blocos “indivisíveis”

Não existe desvio para o meio do bloco

Uma vez que o primeiro comando do bloco é executado, os demais comandos são executados sequencialmente

Arestas ou Arcos: representam o fluxo de controle entre os nós

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

82

82

unesp
BCC
2025

Identifier.c (função *main*)

```
/* 01 */ {
/* 01 */ char  achar;
/* 01 */ int   length, valid_id;
/* 01 */ length = 0;
/* 01 */ printf ("Identificador: ");
/* 01 */ achar = fgetc (stdin);
/* 01 */ valid_id = valid_s(achar);
/* 01 */ if (valid_id)
/* 02 */     length = 1;
/* 03 */ achar = fgetc (stdin);
/* 04 */ while (achar != '\n')
/* 05 */ {
/* 05 */     if (!(valid_f(achar)))
/* 06 */         valid_id = 0;
/* 07 */     length++;
/* 07 */     achar = fgetc (stdin);
/* 07 */ }
/* 08 */ if (valid_id && (length >= 1) && (length < 6) )
/* 09 */     printf ("Valido\n");
/* 10 */ else
/* 10 */     printf ("Invalido\n");
/* 11 */ }
```

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

83

83

unesp
BCC
30/05/2025

Técnica Estrutural

```
/* 01 */ {
/* 01 */     char  achar;
/* 01 */     int   length, valid_id;
/* 01 */     length = 0;
/* 01 */     printf ("Identificador: ");
/* 01 */     achar = fgetc (stdin);
/* 01 */     valid_id = valid_s(achar);
/* 01 */     if (valid_id)
/* 02 */         length = 1;
/* 02 */     achar = fgetc (stdin);
/* 04 */     while (achar != '\n')
/* 05 */     {
/* 05 */         if (!(valid_f(achar)))
/* 06 */             valid_id = 0;
/* 07 */         length++;
/* 07 */         achar = fgetc (stdin);
/* 07 */     }
/* 08 */     if (valid_id && (length >= 1) && (length < 6) )
/* 09 */         printf ("Valido\n");
/* 10 */     else
/* 10 */         printf ("Invalido\n");
/* 11 */ }
```

View Graph - v. 2.0

Nó, arco, caminho
simples (2,3,4,5,6,7)
completo
(1,2,3,4,5,7,4,8,9,11)
fluxo de controle

Grafo de Programa
Gerado pela View-Graph


30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

84

84

Identifier.c (função main)



```
/* 01 */ {
/* 01 */ char  achar;
/* 01 */ int   length, valid_id;
/* 01 */ length = 0;
/* 01 */ printf ("Identificador: ");
/* 01 */ achar = fgetc (stdin);
/* 01 */ valid_id = valid_s(achar);
/* 01 */ if (valid_id)
/* 02 */     length = 1;
/* 03 */ achar = fgetc (stdin);
/* 04 */ while (achar != '\n')
/* 05 */ {
/* 05 */     if (!(valid_f(achar)))
/* 06 */         valid_id = 0;
/* 07 */     length++;
/* 07 */     achar = fgetc (stdin);
/* 07 */ }
/* 08 */ if (valid_id && (length >= 1) && (length < 6) )
/* 09 */     printf ("Valido\n");
/* 10 */ else
/* 10 */     printf ("Invalido\n");
/* 11 */ }
```

Caminho
Não-Executável

Identifier=2
(1,3,4,8,9,11)


30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

85

85

Técnica Estrutural



Critérios da Técnica Estrutural

Baseados em Fluxo de Controle

Todos-Nós, Todas-Arestas e Todos-Caminhos

Baseados em Fluxo de Dados

Critérios de Rapps e Weyuker

Todas-Defs, Todos-Usos, Todos-P-Usos e outros

Critérios Potenciais-Usos (Maldonado)

Todos-Potenciais-Usos, Todos-Potenciais-Usos/DU e outros

Baseados em Complexidade

Critério de McCabe

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

86

86

Técnica Estrutural



Critérios Baseados em Fluxo de Controle

Todos-Nós

1,2,3,4,5,6,7,8,9,10,11

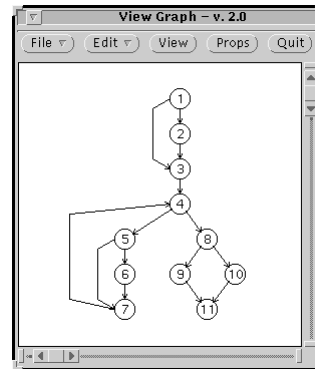
Todos-Arcos

arcos primitivos

<1,2>, <1,3>, <5,6>, <5,7>,

<8,9>, <8,10>

Todos-Caminhos



Grafo de Programa do *identifier*
Gerado pela View-Graph

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

87

87

Técnica Estrutural



Critérios Baseados em Fluxo de Dados

Rapps e Weyuker

Grafo Def-Uso: Grafo de Programa + Definição e Uso de Variáveis

Definição

Atribuição de um valor a uma variável ($a = 1$)

Uso

Predicativo: a variável é utilizada em uma condição
if ($a > 0$)

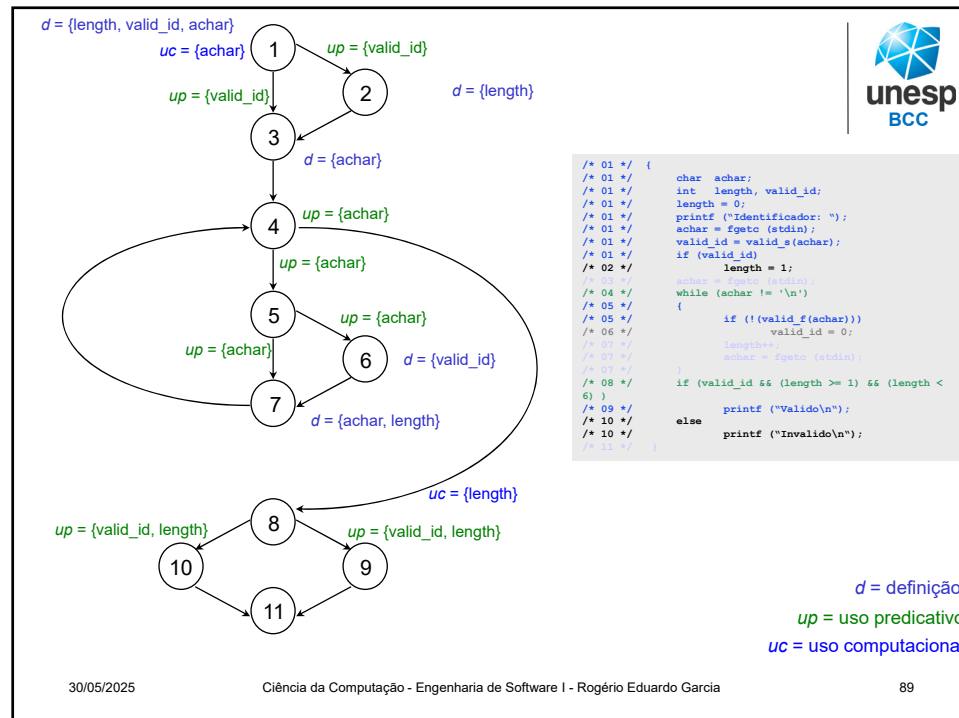
Computacional: a variável é utilizada em uma computação
 $b = a + 1$

30/05/2025

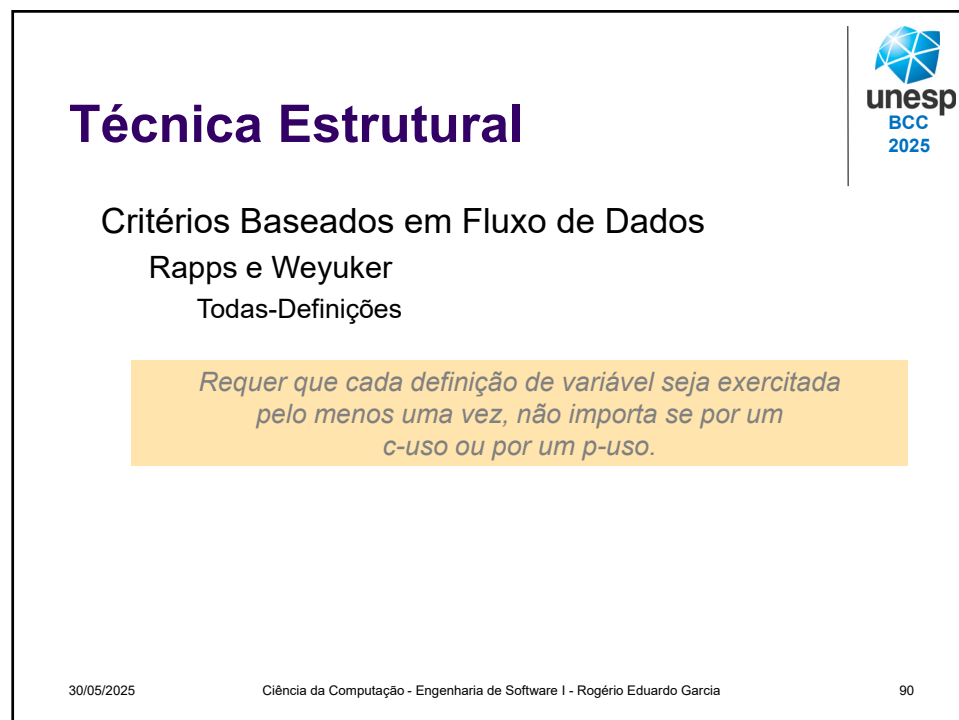
Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

88

88

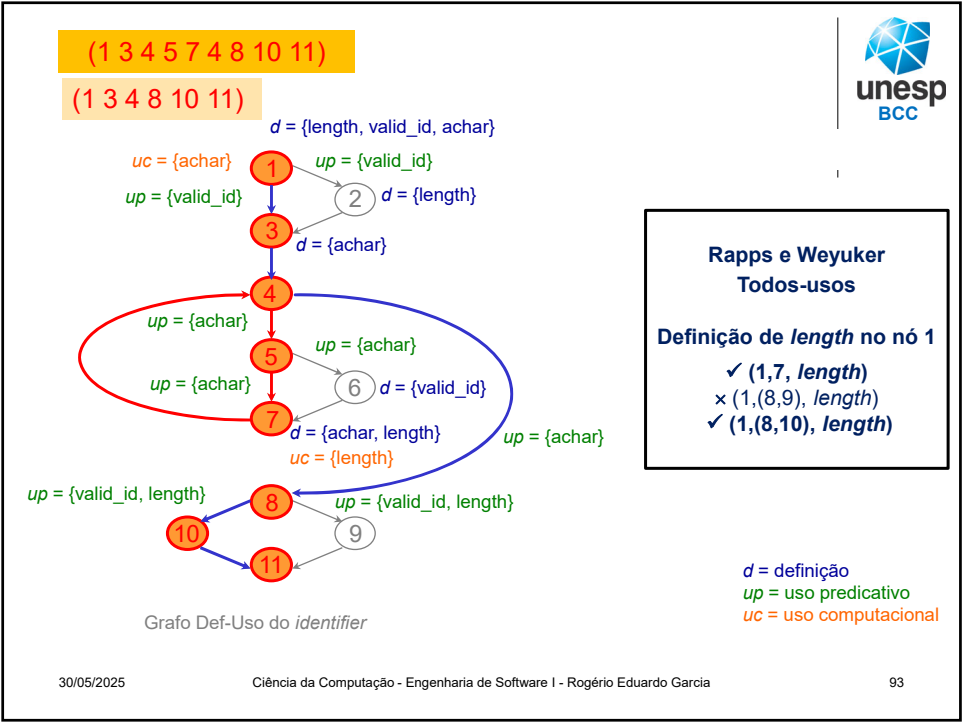


89



90






Técnica Estrutural

Ferramenta *PokeTool*

- Critérios Potenciais-Usos
- Critérios de Rapps e Weyuker
- Outros Critérios Estruturais
 - Todos-Nós, Todos-Arcos
- Linguagem C
- Outras Características
 - Importação de casos de teste
 - Inserção e remoção de casos de teste dinamicamente
 - Casos de teste podem ser habilitados ou desabilitados
 - Geração de relatórios



30/05/2025


Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

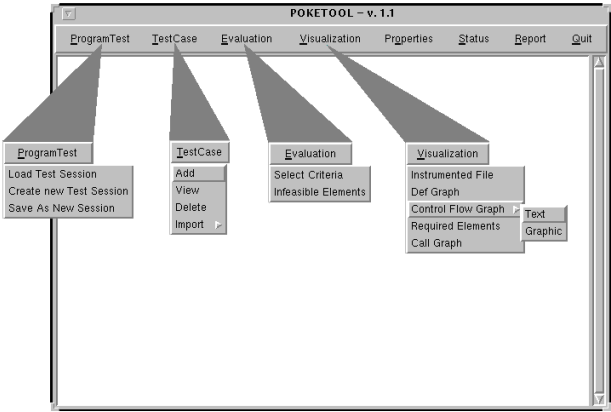
95

95

Técnica Estrutural

PokeTool: Interface Gráfica





30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

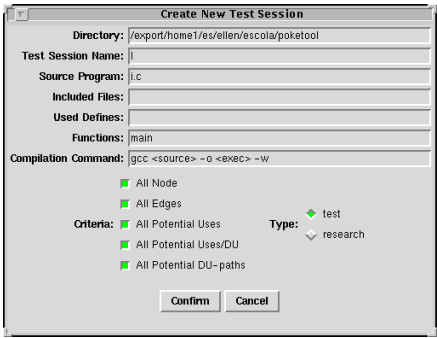
96

96

Técnica Estrutural



PokeTool: Criando uma Sessão de Teste



30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

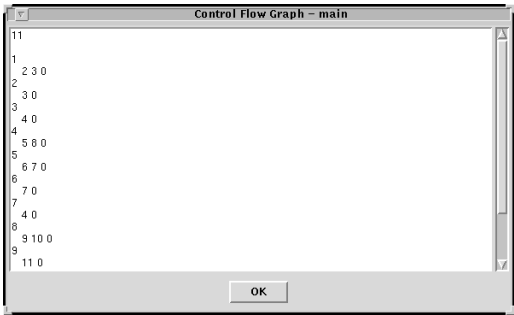
97

97

Técnica Estrutural



PokeTool: Grafo de Programa



30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

98

98

Técnica Estrutural

PokeTool: Elementos Requeridos

Required Elements - main

NO'S DO MODULO main

1 2 3 4 5 6 7 8 9 10
11

ASSOCIACOES REQUERIDAS PELOS CRITERIOS TODOS POT-USOS E POT-USOS/DU

Associacoes requeridas pelo Grafo(1)

1) <1,(6,7),(length)>
2) <1,(1,3),(achar, length, valid_id)>
3) <1,(8,10),(length, valid_id)>
4) <1,(8,10),(valid_id)>
5) <1,(8,9),(length, valid_id)>
6) <1,(8,9),(valid_id)>
7) <1,(7,4),(valid_id)>
8) <1,(5,7),(length, valid_id)>

OK

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

99

99

Técnica Estrutural

- Status após T_0

Status

Directory: /home/auri/identifier/poke

Test Session Name: identifier

Source File: identifier.c

Included Files:

Used Defines:

Compilation Command: gcc <source> -o <exec> -w

Function: main Type: test Total Test Case: 4

Criteria:	Total	Exec	NExec	Infeas	Tot.Cover	Gr.Cover
All Node	11	11	0	0	100	
All Edges	6	6	0	0	100	
All Potential Uses	32	20	12	0	62.50	59.84
All Potential Uses/DU	32	17	15	0	53.12	51.76
All Potential DU-paths	24	11	13	0	45.83	47.50

OK

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

100

$T_0 = \{(a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido})\}$

100

Técnica Estrutural

PokeTool: Relatórios de Teste



30/05/2025

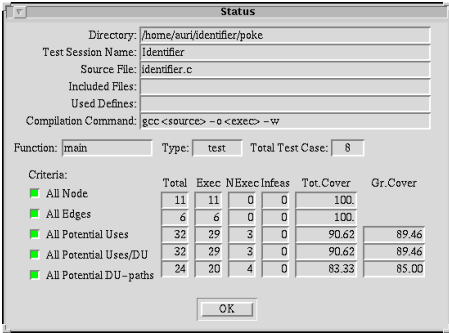
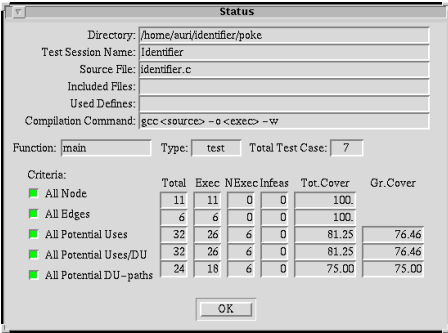
Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

101

101

Técnica Estrutural

- Status após T_1 (a) e T_2 (b)




30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

102

102



Técnica Baseada em Erros


Os requisitos de teste são derivados a partir dos erros mais frequentes cometidos durante o processo de desenvolvimento do software

Critérios da Técnica Baseada em Erros

- Semeadura de Erros
- Teste de Mutação
 - Análise de Mutantes (unidade)
 - Mutação de Interface (integração)

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 103

103



Teste de Mutação

Hipótese do Programador Competente

Programadores experientes escrevem programas corretos ou muito próximos do correto.

Efeito de Acoplamento

Casos de teste capazes de revelar erros simples são tão sensíveis que, implicitamente, também são capazes de revelar erros mais complexos.

30/05/2025 Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia 104

104

unesp
BCC
2025

Teste de Mutação

$$S(t) = P(t) \wedge P(t) \neq M_i(t) \Rightarrow$$
$$P \text{ não contém o erro modelado por } M_i(t)$$

M ₀₁	M ₀₂	M...	M...	M...	M...	M...
M...	M...	M...	M...	M...	M...	M...
M...	M...	M...	P	M...	M...	M...
M...	M...	M...	M...	M...	M...	M...
M...	M...	M...	M...	M...	M...	M...
M...	M...	M...	M...	M...	M...	M _n

Φ(P)

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

105

105

unesp
BCC
2025

Teste de Mutação

Status após a execução de P e M_i

Mutantes Mortos
∃ t / P(t) ≠ M_i(t)

Mutantes Vivos
∀ t ∈ T ⇒ P(t) = M_i(t)

M ₀₁	M ₀₂	M...	M...	M...	M...	M...
M...	M...	M...	M...	M...	M...	M...
M...	M...	M...	P	M...	M...	M...
M...	M...	M...	M...	M...	M...	M...
M...	M...	M...	M...	M...	M...	M...
M...	M...	M...	M...	M...	M...	M _n

Φ(P)

Mutantes Não-Equivalentes

Mutantes Equivalentes

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

106

106

unesp
BCC
2025

Teste de Mutação

The diagram illustrates the mutation testing process. It starts with a program **P** (orange box). Four operators, op_1 , op_2 , op_3 , and op_n , are applied to **P** to generate a set of mutants M_0 through M_{22} . The mutants are represented as colored boxes: blue for non-equivalent mutants, red for equivalent mutants, and orange for live mutants. The mutants are grouped into three categories: **Morto** (Dead), **Não-Equivalentes** (Non-Equivalent), and **Vivos** (Live). The **Morto** category includes M_{17} . The **Não-Equivalentes** category includes M_{18} , M_{19} , and M_{20} . The **Vivos** category includes M_{21} and M_{22} .

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

107

107

unesp
BCC
2025

Análise de Mutantes

Passos da Análise de Mutantes

1- Geração de Mutantes

*Para modelar os desvios sintáticos mais comuns, **operadores de mutação** são aplicados a um programa, transformando-o em programas similares: **mutantes**.*

The diagram shows the process of generating mutants. A program **P** (orange circle) is labeled "Programa em Teste". An arrow labeled "Operadores de Mutação" points to a group of mutants P_1 , P_2 , P_3 , P_4 , and P_n (orange circles). The mutants are labeled "Mutantes".

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

108

108

Análise de Mutantes



Seleção dos operadores de mutação

Abrangente

Capaz de modelar a maior parte dos erros

Pequena cardinalidade

Problemas de custo

Quanto maior o número de operadores utilizados,
maior o número de mutantes gerados

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

109

109

Análise de Mutantes



Exemplo de Mutantes

Mutante Gerado pelo Operador OLAN

```
if (valid_id * (length >= 1) && (length < 6) )  
    printf ("Valido\n");  
else  
    printf ("Invalido\n");
```

Mutante Gerado pelo Operador ORRN


```
if (valid_id && (length >= 1) && (length <= 6) )  
    printf ("Valido\n");  
else  
    printf ("Invalido\n");
```

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

110

110



Análise de Mutantes


Passos da Análise de Mutantes

- 2 - Execução do Programa
 - Execução do programa com os casos de teste
- 3 - Execução dos Mutantes
 - Execução dos mutantes com os casos de teste
 - Mutante morto
 - Mutante vivo
- 4 - Análise dos Mutantes Vivos
 - Mutante equivalente
 - Inclusão de novos casos de teste

Escore de mutação:
$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)}$$

30/05/2025
Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia
111

111



Análise de Mutantes

Ferramenta *Proteum*

- Critério Análise de Mutantes
- Linguagem C
- Outras Características
 - Importação de casos de teste
 - Inserção e remoção de casos de teste dinamicamente
 - Casos de teste podem ser habilitados ou desabilitados
 - Seleção dos operadores a serem utilizados
 - 71 operadores: comandos, operadores, variáveis e constantes
 - Geração de relatórios

30/05/2025
Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia
112

112

Análise de Mutantes

Ferramenta *Proteum*

Interface gráfica


Mais fácil

Constante interação com o testador

Scripts

Possibilitam a condução de uma sessão de teste de modo programado

Domínio dos conceitos de mutação e dos programas que compõem as ferramentas



30/05/2025

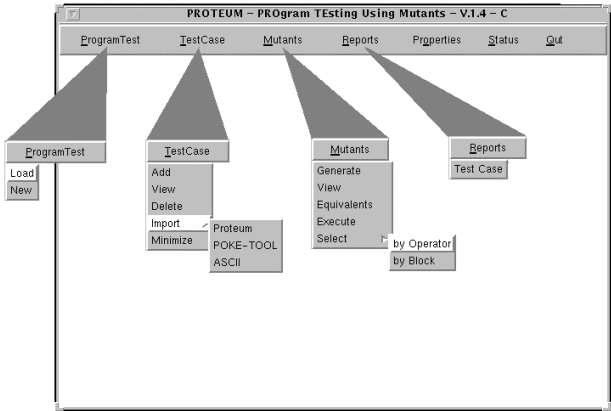
Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia


113

113

Análise de Mutantes

Proteum: Interface Gráfica





30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

114

114

Prof. Dr. Rogério E. Garcia

57

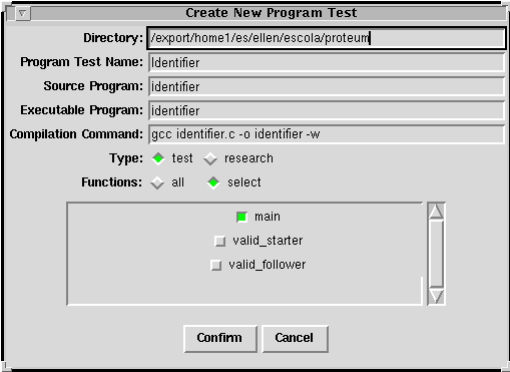
unesp

BCC

2025

Análise de Mutantes

Proteum: Criando uma Sessão de Teste



30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

115

115

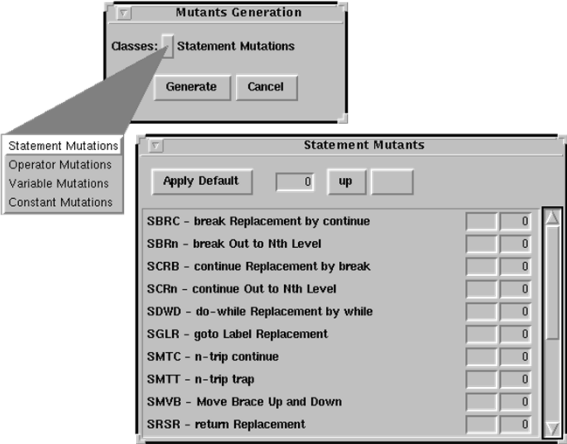
unesp

BCC

2025

Análise de Mutantes

Proteum: Gerando Mutantes



30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

116

116

unesp

BCC

2025

Análise de Mutantes

Status após T_0 (a) e T_2 (b)

Directory: /home/auri/identifer/proteum

Program Test Name: identifier

Source Program: identifier

Executable Program: identifier

Compilation Command: gcc identifier.c -o identifier -w

Type: Test

Test Cases: 4

Total Mutants: 933

Live Mutants: 403

Active Mutants: 933

Anomalous Mutants: 0

Equivalent Mutants: 0

MUTATION SCORE: 0.568

OK

Directory: /home/auri/identifer/proteum

Program Test Name: identifier

Source Program: identifier

Executable Program: identifier

Compilation Command: gcc identifier.c -o identifier -w

Type: Test

Test Cases: 8

Total Mutants: 933

Live Mutants: 371

Active Mutants: 933

Anomalous Mutants: 0

Equivalent Mutants: 0

MUTATION SCORE: 0.602

OK

$T_0 = \{ (a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido}) \}$ $T_1 = T_0 \cup \{ (1\#, \text{Inválido}), (\%, \text{Inválido}), (c, \text{Válido}) \}$ $T_2 = T_1 \cup \{ (\#-\%, \text{Inválido}) \}$

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

117

117

unesp

BCC

2025

Análise de Mutantes

Proteum: Visualização de Mutantes

Mutant: 0

up

down

Type to Show: ☒ Alive ☒ Dead ☒ Anomalous ☒ Equivalent ☐ Inactive

Status: Alive; Active

Operator: Ccrr - Constant for Constant Replacement

Original Program

Mutant Program

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

118

118

Prof. Dr. Rogério E. Garcia

59

unesp

BCC

2025

Análise de Mutantes

Status após T_3 (a) e T_4 (b)

Directory: /home/aur/identfier/proteum

Program Test Name: identifier

Source Program: identifier

Executable Program: identifier

Compilation Command: gcc identifier.c -o identifier -w

Type: Test

Test Cases: 21

Total Mutants: 933

Live Mutants: 64

Active Mutants: 933

Anomalous Mutants: 0

Equivalent Mutants: 78

MUTATION SCORE: 0.925

OK

Directory: /home/aur/identfier/proteum

Program Test Name: identifier

Source Program: identifier

Executable Program: identifier

Compilation Command: gcc identifier.c -o identifier -w

Type: Test

Test Cases: 25

Total Mutants: 933

Live Mutants: 2

Active Mutants: 933

Anomalous Mutants: 0

Equivalent Mutants: 136

MUTATION SCORE: 0.997

OK

(a)

(b)

$$T_3 = T_2 \cup \{(zzz, \text{Válido}), (aA, \text{Válido}), (A1234, \text{Válido}), (ZZZ, \text{Válido}), (AAA, \text{Válido}), (aa09, \text{Válido}), ([, \text{Inválido}), ({, \text{Inválido}), (x/, \text{Inválido}), (x:, \text{Inválido}), (x18, \text{Válido}), (x[, \text{Inválido}), (x{, \text{Inválido})\}$$
$$T_4 = T_3 \cup \{(@, \text{Inválido}), (', \text{Inválido}), (x@, \text{Inválido}), (x', \text{Inválido})\}$$

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

119

119

unesp

BCC

2025

Análise de Mutantes

Mutantes Vivos

Mutante Gerado pelo Operador VTWD

if (valid_id && (length >= 1) && (PRED(length) < 6)

)

printf ("Valido\n");

else

printf ("Invalido\n");

Mutante Gerado pelo Operador ORRN

if (valid_id && (length >= 1) && (length <= 6))

printf ("Valido\n");

else

printf ("Invalido\n");

$t = \{(ABCDEF, \text{Válido})\}$

Saída obtida = Inválido

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

120

120


Prof. Dr. Rogério E. Garcia

60

Identifier.c (função *main*)

Versão Corrigida

```
/* 01 */ {
/* 01 */ char  achar;
/* 01 */ int   length, valid_id;
/* 01 */ length = 0;
/* 01 */ printf ("Digite um possivel identificador\n");
/* 01 */ printf ("seguido por <ENTER>: ");
/* 01 */ achar = fgetc (stdin);
/* 01 */ valid_id = valid_s(achar);
/* 01 */ if (valid_id)
/* 02 */     length = 1;
/* 03 */ achar = fgetc (stdin);
/* 04 */ while (achar != '\n')
/* 05 */ {
/* 05 */     if (!(valid_f(achar)))
/* 06 */         valid_id = 0;
/* 07 */     length++;
/* 07 */     achar = fgetc (stdin);
/* 07 */ }
/* 08 */ if (valid_id && (length >= 1) && (length <= 6) )
/* 09 */     printf ("Valido\n");
/* 10 */ else
/* 10 */     printf ("Invalido\n");
/* 11 */ }
```



30/05/2025


Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

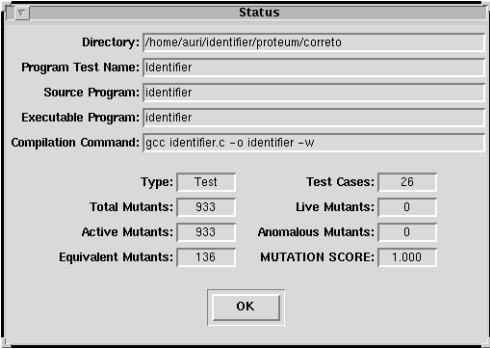
121

121

Análise de Mutantes

Status após T_5 no programa corrigido





$$T_5 = T_4 \cup \{(ABCDEF, \text{Válido})\}$$

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

122

122

Automatização da Atividade de Teste



Ferramentas de Teste

Para a aplicação efetiva de um critério de teste faz-se necessário o uso de ferramentas automatizadas que apoiem a aplicação desse critério.

- Contribuem para reduzir as falhas produzidas pela intervenção humana
 - Aumento da qualidade e produtividade da atividade de teste
 - Aumento da confiabilidade do software
- Facilitam a condução de estudos comparativos entre critérios

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

123

123

Automatização da Atividade de Teste



Critérios Estruturais: Fluxo de Dados

- Asset, Proteste* – programas em Pascal
- xSuds* – programas em C, C++ e Cobol
- Poke-Tool* – programas em C, Cobol e Fortran

Critérios Baseados em Mutação

- Mothra* – programas em Fortran
- Proteum* – programas em C (unidade)
- Proteum/IM* – programas em C (integração)
- Proteum/RS* – especificações


30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

124

124

Automatização da Atividade de Teste



xSuds (Software Understanding & Diagnosis System)

xAtac: teste

xSlice: depuração

xVue: manutenção

xProf: melhoria de performance

xDiff: comparação de código

Estado da Arte X Estado da Prática


30/05/2025

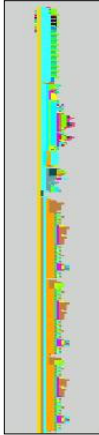
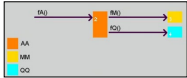
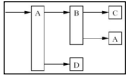
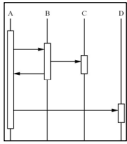
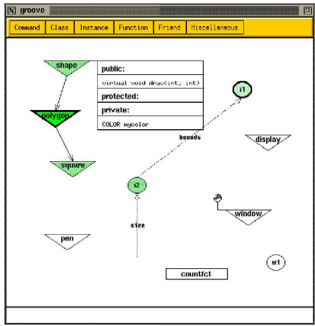
Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

125

125

Orientação a Objetos





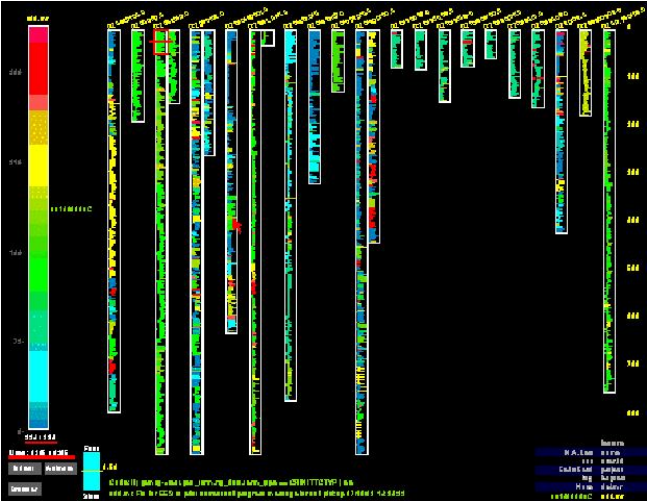
30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

126

126


SeeSoft



30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

127




127

Tarantula

```
mid() {
    int x,y,z,m;
    1: read("Enter 3 numbers: ",x,y,z);
    2: m = x;
    3: if (y<=m)
    4: if (x<y)
    5: m = y;
    6: else if (x<z)
    7: m = z;
    8: else
    9: if (x<y)
    10: m = y;
    11: else if (x<z)
    12: m = z;
    13: print("Middle number is: ",m);
}
```


Casos de Teste						
	3,5,5	1,2,3	2,2,1	3,5,5	5,5,4	2,1,3
1:	●	●	●	●	●	●
2:	●	●	●	●	●	●
3:	●	●	●	●	●	●
4:	●	●	●	●	●	●
5:	●	●	●	●	●	●
6:	●	●	●	●	●	●
7:	●	●	●	●	●	●
8:	●	●	●	●	●	●
9:	●	●	●	●	●	●
10:	●	●	●	●	●	●
11:	●	●	●	●	●	●
12:	●	●	●	●	●	●
13:	●	●	●	●	●	●
Status: Pass/Fail	P	P	P	P	P	F



30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

128



128

Ferramentas *xSuds*



xVue usa heurísticas envolvendo o gráfico do controle, trace de execução e o conhecimento dos mantenedores para ajudar a encontrar trechos de código que implementam requisitos de interesse;

xProf usa o trace de execução para tentar encontrar trechos que correspondam à queda de desempenho do sistema (“gargalos”);

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

129

129

Ferramentas *xSuds*



xAtac combina o grafo de controle e os dados do trace de execução para ajudar a determinar como o código pode ser melhor testado;

xSlice usa o mesmo grafo de controle e os dados do trace de execução para ajudar a encontrar defeitos no código.

xRegress minimiza o conjunto de teste de regressão ajustado de acordo com a cobertura do programa e o custo da execução.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

130

130

unesp

BCC

2025

Ferramentas xProf

File

Tool

Options

Summary

TestCases

Update

GoBack

0

1

12875

25749

38623

51497

64370

77243

90114

case '':
case '%':
TWO_SETHSG(
"YAPC-33E FCIFTVO no escape char preceding delimiter\n");
TV_ERETURN(SYNTAX);
default:
if (curr_char==ecurly || curr_char==clcurly)
{
TWO_SETHSG(
"YAPC-34E FCIFTVO no escape char preceding curly\n");
TV_ERETURN(SYNTAX);
}
}
if (vsizcnt>0 && vsizcnt<WORKSPACE==0)
{
NEWBLOCK();
*cptr+=curr_char;
vsizcnt++;
FSETC();
}
#ifdef TV_DEBUG
#ifdef PC
#else
if (curr_char > -1 && curr_char < 128)
if (isprint(curr_char))
#endif
{
TV_PRINTF(("line 1278 curr_char %c, %X\n", (int)curr_char,
(int)curr_char));
}

xProf

File:
*/yatvo/yafciftv.c

Line:
1143 of 1541

Coverage:
block

Highlighting:
covered

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

131

131

unesp

BCC

2025

Ferramentas xSlice

File

Tool

Options

Summary

TestCases

Update

GoBack

all_passed

all_failed

deselect_all

block coverage summary by testcase over all files

addtagt.1	117 of 9304	1.3%
thsave.1	414 of 9304	4.4%
tputval.1	116 of 9304	1.2%
trn23.1	540 of 9304	5.8%
tapchrM4.1	283 of 9304	3%
atagdhq.1	95 of 9304	1%
thsave2.1	554 of 9304	6%
tputval.1	92 of 9304	1%
trn23.1	667 of 9304	7.2%
tapchrM5.1	283 of 9304	3%
atagdhq2.1	136 of 9304	1.5%
trn23.1	711 of 9304	7.6%
total	17 of 9304	0.2%

xSlice

Coverage:
block

Passed Tests:
2 of 99

Failed Tests:
1 of 99

File

Tool

Options

Summary

TestCases

Update

GoBack

if ('in_path_ptr==' ' && in_path_ptr[1]!='')
{
if (in_path_ptr[2])
{
if (in_path_ptr[2]!='.')
{
TWO_SETHSG(
"ERROR-02E NORM invalid character following right paren\n");
TV_ERROR(BADPATH);
}
}
in_path_ptr+=3;
continue;
}
tptr=in_path_ptr;
while ('tptr')
{
if (isdigit(*tptr)==0) break;
tptr++;
}
while ('tptr==' ') tptr++;
if ('tptr!='')
{
TWO_SETHSG(
"ERROR-03E NORM invalid character following occur no.\n");
TV_ERROR(BADPATH);
}
*occur_ptr=(short)atoi(in_path_ptr);
if (*occur_ptr<0)

xSlice

File:
*/yatvo/yatvcom.c

Line:
82 of 930

Coverage:
block

Highlighting:
covered

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

132

132

CATSDL

Covering a yellow block guarantees the execution of at least five blocks.

Covering a red block guarantees the execution of at least seven blocks.

Move the cursor onto block 7 and the associated SDL specification will be displayed.

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

133

133

Considerações Finais

A atividade de teste é fundamental no processo de desenvolvimento de software

Qualidade do produto

Alto custo da atividade de teste

Desenvolvimento e aplicação de técnicas e critérios de teste

Desenvolvimento e utilização de ferramentas de teste

Estudos teóricos e empíricos para comparar os diversos critérios

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

134

134

Perspectivas



Estratégias de Teste
Teste de Integração
Teste Orientado a Objeto
Teste de Especificação
Teste de Sistemas Reativos
Ambiente Integrado para Teste, Depuração e Manutenção de Software
Teste de programas orientados a aspectos
Teste com o apoio de orientação a aspectos

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

135

135

Leitura Adicional



Livro:

Introdução ao Teste de Software, Delamaro,
Maldonado e Jino, Cap 2, 3, 4 e 6

30/05/2025

Ciência da Computação - Engenharia de Software I - Rogério Eduardo Garcia

136

136