



11ª Semana de Tecnologia Metroferroviária – Fórum Técnico

Associação dos Engenheiros e Arquitetos de Metrô - AEAMESP

IMPORTÂNCIA DA QUALIDADE DE SOFTWARE EM SEGURANÇA CRÍTICA

Introdução

O objetivo deste trabalho, frente ao atual cenário do setor Metroferroviário, é de conscientização e esclarecimento sobre o Controle da Qualidade de Software de Sistemas Sinalização e Controle, demonstrando providências exigíveis a Empresas Fornecedoras e Projetistas destes Sistemas.

No meio acadêmico, são ainda muito discutidas as problemáticas da realização de funções de segurança através de Software. Apesar do tímido número de evidências científicas de que equipamentos baseados em Software se comportam de maneira segura, a prática em diversas áreas demonstra um quadro irreversível de aplicação intensiva desses equipamentos nos mais diversos sistemas, conforme vivenciamos no nosso dia a dia.

A obtenção de um Software Seguro envolve inúmeros fatores, entre eles a experiência e o preparo de toda a equipe envolvida no Processo, o uso de técnicas e ferramentas apropriadas, uma criteriosa Análise de Segurança Independente e, dentre os atributos da Qualidade de Software, em especial sua Confiabilidade.

Confiabilidade de Software

A Confiabilidade de Software pode ser definida como a “probabilidade de operação livre de falhas por um período de tempo especificado em um determinado ambiente” [ANSI 1991]. A Confiabilidade é um dos atributos da Qualidade de Software, uma propriedade multidimensional que inclui outros fatores de satisfação do cliente tais como funcionalidade, facilidade de uso, desempenho, capacidade, documentação, facilidades de suporte, instalação e manutenção [Grad 1987, Grad 1992].

A Confiabilidade de Software é geralmente aceita como o fator principal da Qualidade de Software, pois quantifica as falhas de Software – que podem tirar de operação um poderoso sistema ou até mesmo trazer riscos às pessoas e patrimônio [Lyu 1996].



Os métodos mais respeitados e evoluídos para a avaliação da confiabilidade de software, tal como descrito em “Software Reliability Engineering” [Musa 1999], são baseados na observação de falhas do produto terminado, sustentada por análise estatística. Estes métodos são a única forma de medir a confiabilidade do produto, pois são baseadas na observação direta de seu comportamento. As importantes desvantagens destas abordagens são [Dale 1990]:

- estão preocupadas puramente com a medida da confiabilidade, enquanto outras informações sobre, por exemplo, os métodos que foram utilizados no desenvolvimento do produto, são ignoradas; e
- só podem ser utilizadas quando o produto final existe, portanto sua aplicação só pode ser iniciada a partir da obtenção de dados durante a fase de testes do desenvolvimento.

Uma alternativa pesquisada para a obtenção de Software de alta confiabilidade é a especificação e análise dos Processos de Desenvolvimento de Software adotados em um projeto, em conformidade com as normas técnicas adequadas a aplicação, além de modelos de ampla aceitação.

Adotando-se uma abordagem deste tipo, a avaliação da confiabilidade deve ser realizada sobre os produtos intermediários de cada fase do processo de desenvolvimento, antes de se obter o produto final.

Segurança de Software (SW Safety)

A Segurança de um sistema $S(t)$ é a probabilidade de um sistema efetuar suas operações de forma correta, ou descontinuar seu funcionamento de forma a não comprometer a operação de outros sistemas ou comprometer a Segurança [Johnson 1989]. As técnicas de “falha segura” (fail-safe) têm sido empregadas desde em sistemas eletrônicos convencionais e até mesmo nas principais rotinas de diagnóstico de funções de Software e componentes de Hardware, bem como nos conceitos de programação defensiva.

Especificação de Requisitos Técnicos de Software

A Especificação Técnica de Software abordada a seguir foi elaborada, a princípio, tendo em vista um horizonte de aplicação da tecnologia “Communications Based Train Control” (CBTC) na

implantação e modernização de Sistemas de Sinalização e Controle. O emprego desta nova tecnologia traz desafios tais como:

- uso intensivo de Software para a realização de funções de Tempo Real e Segurança Crítica;
- fatores de aumento natural do risco e a conseqüente necessidade de sua mitigação através da tecnologia, tais como a diminuição do intervalo entre trens (“headway”), operação automática sem condutor (“driverless”), funções de diagnóstico e recuperação remotas;
- transmissão intensiva por meio aberto de sinais vitais, trazendo preocupações típicas da Segurança da Informação (“security”), além do envelhecimento de informações de tempo real, que devem ser tratadas pelo Software de comunicação;
- determinação da posição das composições e distâncias percorridas utilizando múltiplos sensores, e tratamento por algoritmos que incluem informações sobre margens de segurança para determinação e cumprimento de pontos de parada com segurança;
- emprego limitado de hardware intrinsecamente “fail-safe” e componentes com modos de falha bem definidos; e
- os conceitos consagrados de Segurança de um Sistema de Sinalização e Controle devem ser implementados de forma completa e claramente verificáveis [IEEE 2005].

No entanto, devido a incorporação da experiência obtida pelo METRÔ-SP no projeto e implantação dos sistemas de Sinalização e Controle, somada a estudos realizados em conjunto com a Fundação para o Desenvolvimento Tecnológico da Engenharia - FDTE, foi considerado adequado o emprego desta metodologia para a aquisição de Software do Sistema de Sinalização e Controle de Trens – Estações e Vias da Extensão da Linha 2 – Verde, trecho Ana Rosa – Ipiranga, mesmo tratando-se de um Sistema Microprocessado Convencional, e dentre seus principais requisitos, serão enfatizados:

❖ **Conformidade com CMMi**

A Capacidade do Processo [CMU/SEI 2002] estabelece o grau de refinamento e institucionalização com que o processo é realizado em uma organização, em cinco níveis discretos, em sua representação por estágios, conforme representado na Figura 1.

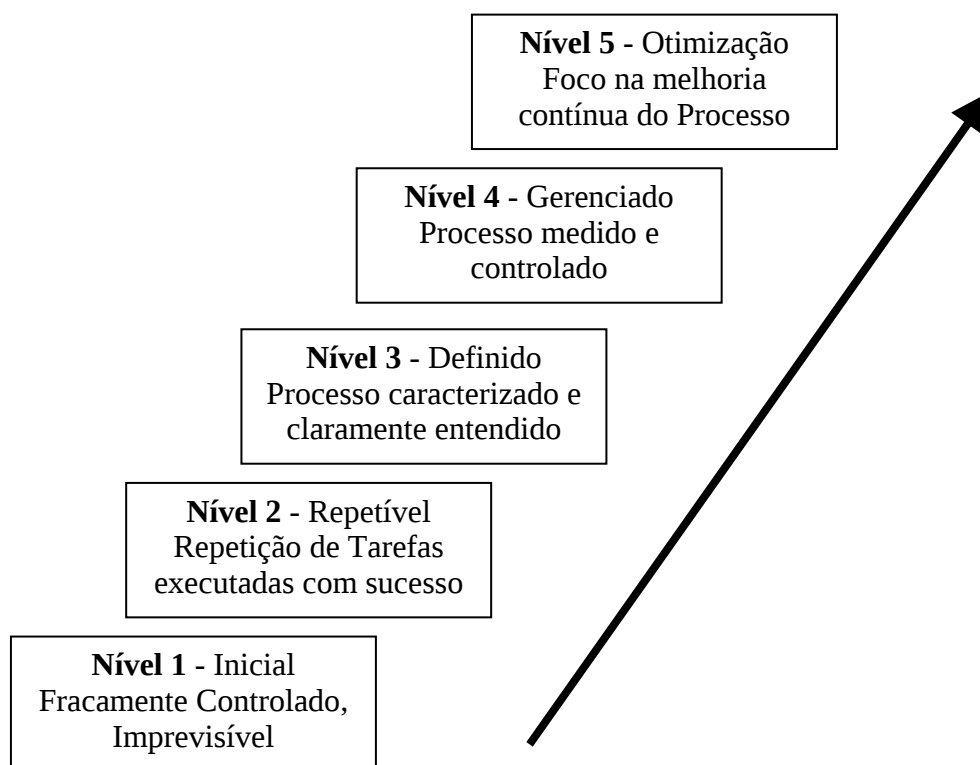


Figura 1 - Estágios do CMMi

Os atributos de processo pertinentes ao nível 2 do CMMi, pelo qual estamos iniciando a utilização desta metodologia, ilustrados na Figura 2, podem ser assim resumidos:

- o Processo é Executado;
- o Processo atinge seus resultados definidos; e
- os produtos de trabalho do Processo são gerenciados.

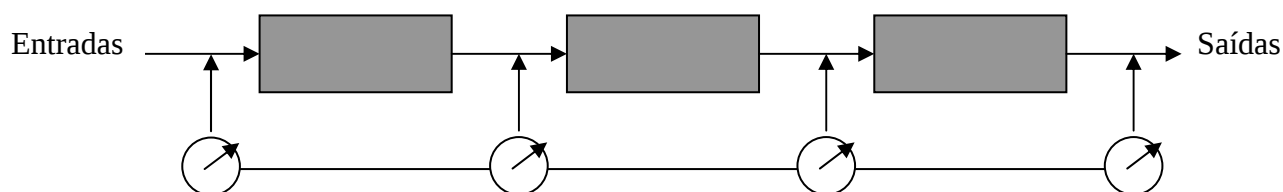


Figura 2 - CMMi nível 2 - Visibilidade

A Especificação Técnica de Software apresenta o seguinte requisito:

A Contratada deve adotar a metodologia estabelecida no “Capability Maturity Model Integration for Systems Engineering and Software Engineering” (CMMI-SE/SW), versão 1.1 ou superior, do “Software Engineering Institute / Carnegie Mellon University” (SEI/CMU) para todo o Software fornecido, sendo que:



Deve ser adotado CMMI-SE/SW nível dois (2) ou superior, na representação em estágios “Staged Representation”;

A Contratada deve comprovar pelo “Standard CMMI Appraisal Method for Process Improvement” (SCAMPI) [CMU/SEI 2001], o emprego desta metodologia por toda equipe envolvida no projeto.

❖ Nível de Integridade de Segurança (SIL)

A Especificação Técnica de Software apresenta o seguinte requisito com relação ao Nível de Integridade de Segurança de Software, que juntamente com os parâmetros de Segurança do Hardware irão determinar a Segurança do Sistema, traduzido em valores na Figura 3.

Os conceitos de “Fail Safe” devem ser utilizados no Software relacionado às funções de Segurança Crítica ou Vitais. Deve atender ao “Software Safety Integrity Level” (SIL) quatro (4), conforme a norma IEC 61508 “Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems”.

SIL	Falhas inseguras / hora
4	$10^{-9} < \lambda_u < 10^{-8}$
3	$10^{-8} < \lambda_u < 10^{-7}$
2	$10^{-7} < \lambda_u < 10^{-6}$
1	$10^{-6} < \lambda_u < 10^{-5}$

Figura 3 – Nível de Integridade de Segurança [IEC 1999]

❖ Conformidade com a Norma CENELEC 50128

Um importante requisito na Especificação Técnica de Software é a conformidade com a Norma CENELEC 50128 [CENELEC 2001], específica para o Software aplicado a Sistemas de Sinalização e Controle:

Devem ser atendidas as recomendações da norma CENELEC 50128 “Railway applications – Communications, Signaling and Processing Systems – Software for Railway Control and Protection Systems”.

Conforme podemos observar no diagrama seguinte, presente na Figura 4, as atividades específicas para a obtenção do Software estão vinculadas às especificações do Sistema onde ele é empregado.

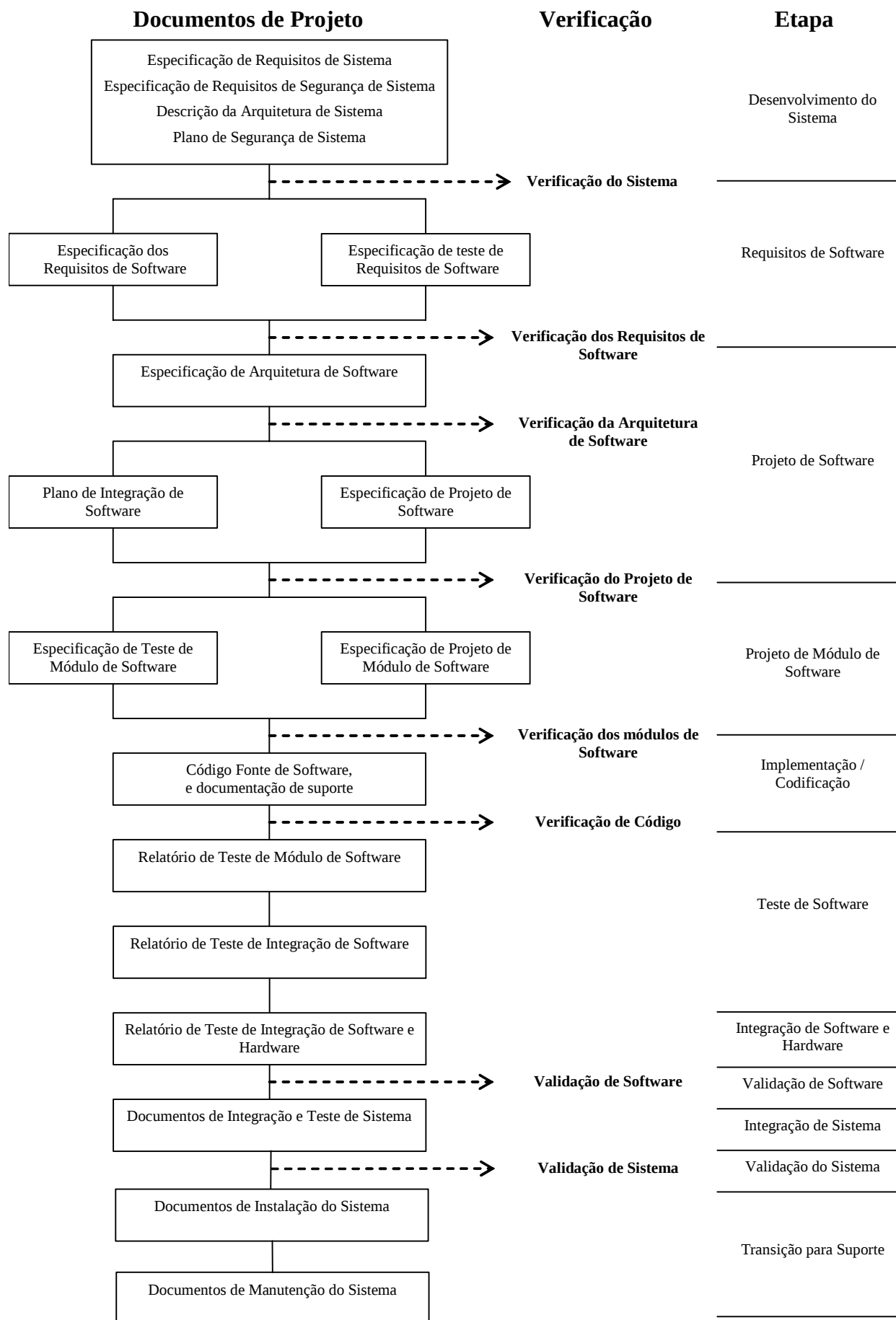


Figura 4 - Ciclo de Vida e Artefatos [CENELEC 2001]

❖ Planejamento de Software

Antes do início do desenvolvimento do Software, foi detectada a necessidade de solicitar à empresa fornecedora e projetista a elaboração de Planos específicos aos processos de obtenção de Software com o nível de Segurança adequado à aplicação no sistema de Sinalização e Controle:

- Plano de Gerenciamento de Software;
- Plano de Garantia da Qualidade de Software;
- Plano de Gerência de Configuração de Software;
- Plano de Verificação de Software de Aplicação Crítica e de Segurança; e
- Plano de Validação de Software.

Estes documentos deverão ser submetidos à aprovação, incluindo eventuais alterações ao longo do fornecimento.

❖ Ciclo de Vida

A Especificação Técnica de Software apresenta, ainda, uma descrição das fases de um Ciclo de Vida do Software de referência, listadas a seguir:

- Especificação de Requisitos de Software;
- Arquitetura de Software;
- Projeto de Software;
- Implementação de Software;
- Integração e Testes de Software;
- Validação de Software;
- Aceitação de Software; e
- Operação e Manutenção.

Conclusões

Este trabalho tem por objetivo obter uma maior garantia do nível de Segurança do Software e, ainda, a redução do re-trabalho e conseqüente otimização dos recursos empregados, possibilitando também um maior controle sobre o processo e as tecnologias em implantação.

Não deixando de observar as limitações impostas pelo estágio de desenvolvimento atual da indústria nacional, é importante vislumbrar esta oportunidade e necessidade de progresso,



aproveitando a experiência e capacidade de inovação, e incorporando a iniciativa dos maiores consumidores de Software mundiais e a produção científico-acadêmica disponível nesta área.

Referência Bibliográfica

- [CMU/SEI 2002] Carnegie Mellon University Software Engineering Institute: “CMMI-SE/SW – Capability Maturity Model Integration (CMMI), Version 1.1: CMMI for Systems Engineering and Software Engineering, Staged Representation”, CMU/SEI-2002-TR-002, 2002.
- [CMU/SEI 2001] Carnegie Mellon University Software Engineering Institute: “SCAMPI – Standard CMMI Appraisal Method for Process Improvement Version 1.1: Method Definition Document”, CMU/SEI-2001-HB-001, 2001.
- [CENELEC 2001] European Committee for Electrotechnical Standardization: “Railway Applications – Communications, Signaling and Processing Systems – Software for Railway Control and Protection Systems”, BS-EN 50128, 2001.
- [Dale 1990] Dale, C.; Paul Rook (ed.) – “Software Reliability Handbook” – Elsevier Applied Science, 1990.
- [Johnson 1989] Barry W. Johnson – “Design and Analyses of fault-Tolerant Digital Systems” – Addison-Wesley, 1989.
- [Lyu 1996] Michael R. Lyu (ed.) – “Handbook of Software Reliability Engineering”, McGraw-Hill, 1996.
- [Musa 1999] Musa J.D. – “Software Reliability Engineering”, McGraw-Hill, 1999.
- [ANSI 1991] ANSI/IEEE, “Standard Glossary of Software Engineering Terminology”, STD-729-1991, ANSI/IEEE, 1991.
- [GRAD 1987] Grady, Robert B.; Caswell, Deborah L.- “Software Metrics: Establishing a Company-Wide Program.” Prentice Hall, 1987.
- [GRAD 1992] Grady, Robert B. - “Practical Software Metrics for Project Management and Process Improvement”. Prentice Hall, 1992.
- [IEEE 2005] IEEE Standard for Communications-Based Train Control (CBTC) Performance and Functional Requirements, Institute of Electrical and Electronics Engineers, IEEE 1474-1, 2005.
- [IEC 1999] Functional safety of electrical/electronic/programmable electronic safety-related systems - General requirements, IEC 61508-1, 1999.

Eng. Christian Becker Bueno de Abreu

e-mail: cbecker@metrosp.com.br – tel. 55-11-3371-7222

Engenheiro de Projeto da Companhia do Metropolitano de São Paulo – METRÔ

Departamento de Projeto de Sistemas de Controle e Comunicações – GEP/EPS

Prof. Dr. Paulo Sérgio Cugnasca

Escola Politécnica da Universidade de São Paulo (EPUSP)

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)

Grupo de Análise de Segurança (GAS)

11ª Semana de Tecnologia Metroferroviária

Importância da Qualidade de Software em Segurança Crítica

Eng. Christian *Becker* Bueno de Abreu

Prof. Dr. Paulo Sérgio Cugnasca

```

89 void IniciaBAC (void)
90 {
91     DgnBAC = 0;
92     UFTxBAC = 0;
93     PFTxBAC = 0;
94     AFTxBAC = 0;
95     QFTxBAC = 0;
96     UFRxBAC = 0;
97     PFRxBAC = 0;
98     QFRxBAC = 0;
99     FlagTxBAC = 0;
100     FlagEstadoBAC = 0;
101     FlagCpcBAC = 0;
102     FalhasBAC = 0;
103     NrtxBAC = MAXRETXBAC;
104     SemaforoTxBAC = 0x0F;
105     TimerTimeoutBAC = TimerRelogio3;
106     TimerJanelaBAC = TimerRelogio3;
107     ProgramaRxBAC = 0;
108     InsereTarBAC(0);
109
110     #if MESTREBAC == 1
111     #endif
112     return;
113 }
114
115 void TestaFilaBAC (void)
116 {
117     Diagnostico[0x44]++;
118     disable();
119     if ((PFRxBAC >= MAXFRXBAC) ||
120         (UFRxBAC >= MAXFRXBAC) ||
121         (QFRxBAC >= MAXFRXBAC) ||
122         (PFTxBAC >= MAXFTXBAC) ||
123         (UFTxBAC >= MAXFTXBAC) ||
124         (QFTxBAC >= MAXFTXBAC))
125     {
126         /* Mensagem na fila de recepcao. */
127         if (QFRxBAC > 0)
128         {
129             /* Sim. Copia a mensagem para o ponteiro. */
130             i = 0;
131             while (i < FrxBAC[PFRxBAC].m[0])
132             {
133                 msi[i] = FrxBAC[PFRxBAC].m[i+1];
134                 i++;
135             }
136             /* [02] */
137             /* Atualiza indices da fila de recepcao. */
138             disable();
139             if (PFRxBAC >= (MAXFRXBAC-1))
140                 PFRxBAC = 0;
141             else
142                 PFRxBAC ++;
143             QFRxBAC --;
144             if (QFRxBAC < 0)
145                 QFRxBAC = MAXFTXBAC-1;
146             return(i);
147         }
148         /* [01] */
149         if ((SemaforoTxBAC == 0) && (QFTxBAC < MAXFTXBAC) &&
150             (n < MAXBFTXBAC))
151         {
152             /* Copia a mensagem para a fila. */
153             i = 0;
154             while (i < n)
155             {
156                 /* ... */
157             }
158             /* ... */
159             return;
160         }
161     }
162     BYTE EnviaMsgBAC (BYTE *msg, BYTE n)
163     {
164         BYTE i;
165
166         /* [01] */
167         if ((SemaforoTxBAC == 0) && (QFTxBAC < MAXFTXBAC) &&
168             (n < MAXBFTXBAC))
169         {
170             /* Copia a mensagem para a fila. */
171             i = 0;
172             while (i < n)
173             {
174                 /* ... */
175             }
176             /* ... */
177             return(0);
178         }
179         return(i);
180     }
181 }
182
183 void EnviaAckBAC (void)
184 {
185     TxMsgAckBAC();
186     FlagTxBAC = 2;
187     TimerTimeoutBAC = TimerRelogio3;
188     return;
189 }
190
191 void EnviaNackBAC (void)
192 {
193     /* ... */
194 }

```

```
89 void IniciaBAC (void)
```

```
90 {
91     1 Diagnostico[0x07] ++;
92     1 UFTxBAC = 0;
93     1 PFTxBAC = 0;
94     1 AFTxBAC = 0;
95     1 QFTxBAC = 0;
96     1 UFRxBAC = 0;
97     1 PFRxBAC = 0;
98     1 QFRxBAC = 0;
99     1 FlagTxBAC = 0;
100    1 FlagEstado = 0;
101    1 FlagCiclo = 0;
102    1 FalhaCiclo = 0;
103    1 NrtxBAC = 0;
104    1 TimerRelogio3 = 0;
105    1 TimerRelogio2 = 0;
106    1 TimerRelogio1 = 0;
107    1 TimerRelogio0 = 0;
108    1 TimerRelogio4 = 0;
109    1 TimerRelogio5 = 0;
110    1 TimerRelogio6 = 0;
```

```
113    1
114    1
141
142
143    1
144    1
145    1
146    1
147    1
148    1
149    1
150    1
151    1
152    1
153    2
154    2
155    2
156    2
157    2
158    1
159    1
160    1
161    1
162    1
163    1
164    1
165    1
166    1
167    1
168    1
169    1
170    1
171    1
172    1
173    1
174    1
175    1
176    1
177    1
178    1
179    1
180    1
181    1
182    1
183    1
184    1
185    1
186    1
187    1
188    1
189    1
190    1
191    1
192    1
193    1
194    1
195    1
196    1
197    1
198    2
199    2
200    2
```

Software - Definições

Natureza do Software

???

Todas

ção de um

[NBR ISO 9000-3]

```
277    1
280    1
281    1
282    1
283    1
284    1
285    1
286    1
287    1
288    1
289    1
290    1
291    1
292    1
293    1
294    1
295    1
296    1
297    1
298    1
299    1
300    1
301    1
302    1
303    1
304    1
305    1
306    1
307    1
308    1
309    1
310    1
311    1
312    1
313    1
314    1
315    1
316    1
317    1
318    1
319    1
320    1
321    1
322    1
323    1
324    1
325    1
326    1
327    1
328    1
329    1
330    1
331    1
332    1
333    1
```

```
return(0);
void EnviaAckBAC (void)
{
    TxMsgAckBAC();
    FlagTxBAC = 2;
    TimerTimeoutBAC = TimerRelogio3;
    return;
}
void EnviaNackBAC (void)
```



```
89 void IniciaBAC (void)
```

```
90 {
91     1 Diagnostico[0x07] ++;
92     1 UFTxBAC = 0;
93     1 PFTxBAC = 0;
94     1 AFTxBAC = 0;
95     1 QFTxBAC = 0;
96     1 UFRxBAC = 0;
97     1 PFRxBAC = 0;
98     1 QFRxBAC = 0;
99     1 FlagTxBAC = 0;
100    1 FlagEstadoBAC = 0;
101    1 FlagCicloBAC = 0;
102    1 FalhaBAC = 0;
103    1 NrtxBAC = MAXRETXBAC;
104    1 TimerJanelaBAC = TimerRelogio2;
105    1 ProgramaRS232C();
106    1 InsereTarefa(14, NrtxBAC);
107    1
108    1 #if MESTREBAC == 1
109    1 #endif
110    1 }
111
112 void TestaFilaBAC (void)
113 {
114     1 Diagnostico[0x44]++;
115     1 disable();
116     1 if ((PFRxBAC >= MAXFRXBAC) ||
117     1     (UFRxBAC >= MAXFRXBAC) ||
118     1     (QFRxBAC >= MAXFRXBAC) ||
119     1     (PFTxBAC >= MAXFTXBAC) ||
120     1     (UFTxBAC >= MAXFTXBAC) ||
121     1     (QFTxBAC >= MAXFTXBAC))
122     1 {
123         2 module();
124         2 return;
125     1 }
126
127 BYTE EnviaMsgBAC (BYTE *msg, BYTE n)
128 {
129     1 BYTE i;
130     1
131     1 /* [01] */
132     1 if ((SemaforoTxBAC == 0) && (QFTxBAC < MAXFTXBAC) &&
133     1     (n < MAXBFTXBAC))
134     1 {
135         2 /* Copia a mensagem para a fila. */
136         2 i = 0;
137         2 while (i < n)
138         2 {
```

Software - Definições

“Software”:

Criação intelectual compreendendo os programas, procedimentos, regras e qualquer documentação correlata de um sistema de processamento de dados

NOTA: O “software” não depende do meio no qual é registrado.

Desenvolvimento:

Todas as atividades a serem realizadas para a criação de um produto de Software.

[NBR ISO 9000-3]

Qualidade & Segurança - Definições

Qualidade - indicadores

- **Confiabilidade**
- Funcionalidade
- Facilidade de uso
- Desempenho
- Capacidade
- Documentação
- Suporte, Instalação e Manutenção

Segurança - S(t)

Probabilidade de um sistema efetuar suas operações de forma correta, sem desconectar seu funcionamento de forma a não comprometer a operação de outros sistemas ou comprometer a Segurança

```
89 void IniciaBAC (void)
90 {
91     1 DiagBAC = 0;
92     1 UFTxBAC = 0;
93     1 PFTxBAC = 0;
94     1 AFTxBAC = 0;
95     1 QFTxBAC = 0;
96     1 UFRxBAC = 0;
97     1 PFRxBAC = 0;
98     1 QFRxBAC = 0;
99     1 FlagTxBAC = 0;
100     1 FlagCpcBAC = 0;
101     1 FalhasBAC = 0;
102     1 NrtxBAC = MAXRETXBAC;
103     1 SemaforoTxBAC = 0x0F;
104     1 TimerTimeoutBAC = TimerRelogio3;
105     1 TimerJanelaBAC = TimerRelogio2;
106     1 ProgramaS2316(
107     1     1 ^ NIL ^ NIL));
108     1
109     1 #if MESTREBAC == 1
110     1
111     1 #endif
112     1
113     1 void TestaFilaBAC (void)
114     1
115     1 {
116     1     1 disable();
117     1     1 if ((PFRxBAC >= MAXPFRxBAC) ||
118     1     1     1 (QFRxBAC >= MAXQFRxBAC) ||
119     1     1     1 (PFTxBAC >= MAXPFTxBAC) ||
120     1     1     1 (QFTxBAC >= MAXQFTxBAC))
121     1     1 {
122     1     1     1 /* Copia a mensagem para a fila de recepcao. */
123     1     1     1 i = 0;
124     1     1     1 while (i < n)
125     1     1     1 {
126     1     1     1     1 msg[i] = FrxBAC[PFRxBAC].m[i+1];
127     1     1     1     1 i++;
128     1     1     1 }
129     1     1     1 /* [02] */
130     1     1     1 /* Atualiza indices da fila de recepcao. */
131     1     1     1 disable();
132     1     1     1 if (PFRxBAC >= (MAXFRXBAC-1))
133     1     1     1     1 PFRxBAC = 0;
134     1     1     1 else
135     1     1     1     1 PFRxBAC++;
136     1     1     1 QFRxBAC--;
137     1     1     1 enable();
138     1     1     1 return(i);
139     1     1 }
140     1     1 return(0);
141     1 }
142     1 void EnviaAckBAC (void)
143     1 {
144     1     1 TxMsgAckBAC();
145     1     1 FlagTxBAC = 2;
146     1     1 TimerTimeoutBAC = TimerRelogio3;
147     1     1 return;
148     1 }
149     1 void EnviaNackBAC (void)
150     1 {
151     1     1 TxMsgAckBAC();
152     1     1 FlagTxBAC = 2;
153     1     1 TimerTimeoutBAC = TimerRelogio3;
154     1     1 return;
155     1 }
156     1 }
157     1 }
158     1 }
159     1 }
160     1 }
```

```
202     3 FrxBAC[UFrxBAC].m[i+1] = 0;
203     3 i = 303;
204     3 }
205     2
206     2 FrxBAC[UFrxBAC].m[i+1] = 0;
207     2
208     2
209     2 /* Atualiza indices da fila de transmissao. */
210     2 if (PFTxBAC >= (MAXFTXBAC-1))
211     2     2 PFTxBAC = 0;
212     2 else
213     2     2 PFTxBAC++;
214     2 QFTxBAC--;
215     2 enable();
216     2 return(TRUE);
217     2 }
218     2 return(FALSE);
219     2 }
220     2 }
221     2 }
222     2 }
223     2 }
224     2 }
225     2 }
226     2 }
227     2 }
228     2 }
229     2 }
230     1 BYTE 0;
231     1 BYTE 1;
232     1 }
233     1 if (QFRxBAC > 0)
234     1 {
235     1     1 /* Copia a mensagem para o primeiro. */
236     1     1 i = 0;
237     1     1 while (i < FrxBAC[PFRxBAC].m[0])
238     1     1 {
239     1     1     1 msg[i] = FrxBAC[PFRxBAC].m[i+1];
240     1     1     1 i++;
241     1     1 }
242     1     1 /* [02] */
243     1     1 /* Atualiza indices da fila de recepcao. */
244     1     1 disable();
245     1     1 if (PFRxBAC >= (MAXFRXBAC-1))
246     1     1     1 PFRxBAC = 0;
247     1     1 else
248     1     1     1 PFRxBAC++;
249     1     1 QFRxBAC--;
250     1     1 enable();
251     1     1 return(i);
252     1     1 }
253     1     1 return(0);
254     1 }
255     1 void EnviaAckBAC (void)
256     1 {
257     1     1 TxMsgAckBAC();
258     1     1 FlagTxBAC = 2;
259     1     1 TimerTimeoutBAC = TimerRelogio3;
260     1     1 return;
261     1 }
262     1 void EnviaNackBAC (void)
263     1 {
264     1     1 TxMsgAckBAC();
265     1     1 FlagTxBAC = 2;
266     1     1 TimerTimeoutBAC = TimerRelogio3;
267     1     1 return;
268     1 }
269     1 }
```

Qualidade $\cap$ Segurança - Confiabilidade

QUALIDADE

CONFIABILIDADE

Confiabilidade - $R(t)$

Probabilidade de **operação livre de falhas** por um período de tempo especificado em um determinado ambiente

$$MTTF = \int_0^{\infty} R(t) \cdot dt$$

```

89 void IniciaBAC (void)
90 {
91     1  UFRxBAC = 0;
92     1  PFRxBAC = 0;
93     1  AFRxBAC = 0;
94     1  QFRxBAC = 0;
95     1  UFRxBAC = 0;
96     1  PFRxBAC = 0;
97     1  QFRxBAC = 0;
98     1  FlagTxBAC = 0;
99     1  FlagTxBAC = 0;
100
101     1  enable();
102     1  return;
103 }
104
105 BYTE EnviaMsgBAC (BYTE *msg, BYTE n)
106 {
107     1  BYTE i;
108     1
109     1  /* [01] */
110     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
111         1  (n < MAXBFTXBAC))
112     1  {
113         2  /* Copia a mensagem para a fila. */
114         2  i = 0;
115         2  while (i < n)
116         2  {
117             3  m[i] = msg[i];
118             3  i++;
119         }
120         2  /* Sim. Copia a mensagem para o ponteiro. */
121         2  i = 0;
122         2  while (i < n)
123         2  {
124             3  m[i] = msg[i];
125             3  i++;
126         }
127         2  /* [02] */
128         2  /* Atualiza indices da fila de recepcao. */
129         2  disable();
130         2  if (PFRxBAC >= (MAXFRXBAC-1))
131         2  {
132             3  PFRxBAC = 0;
133         }
134         2  else
135         2  {
136             3  PFRxBAC ++;
137             3  QFRxBAC --;
138             3  enable();
139         }
140         2  return(0);
141     }
142     1  return(1);
143 }
144
145 void EnviaAckBAC (void)
146 {
147     1  TxMsgAckBAC();
148     1  FlagTxBAC = 2;
149     1  TimerTimeoutBAC = TimerRelogio3;
150     1  return;
151 }
152
153 void EnviaNackBAC (void)
154 {
155     1  TxMsgAckBAC();
156     1  FlagTxBAC = 2;
157     1  TimerTimeoutBAC = TimerRelogio3;
158     1  return;
159 }
160
161 void EnviaMsgBAC (BYTE *msg, BYTE n)
162 {
163     1  BYTE i;
164     1
165     1  /* [01] */
166     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
167         1  (n < MAXBFTXBAC))
168     1  {
169         2  /* Copia a mensagem para a fila. */
170         2  i = 0;
171         2  while (i < n)
172         2  {
173             3  m[i] = msg[i];
174             3  i++;
175         }
176         2  /* Sim. Copia a mensagem para o ponteiro. */
177         2  i = 0;
178         2  while (i < n)
179         2  {
180             3  m[i] = msg[i];
181             3  i++;
182         }
183         2  /* [02] */
184         2  /* Atualiza indices da fila de recepcao. */
185         2  disable();
186         2  if (PFRxBAC >= (MAXFRXBAC-1))
187         2  {
188             3  PFRxBAC = 0;
189         }
190         2  else
191         2  {
192             3  PFRxBAC ++;
193             3  QFRxBAC --;
194             3  enable();
195         }
196         2  return(0);
197     }
198     1  return(1);
199 }
200
201 void EnviaAckBAC (void)
202 {
203     1  TxMsgAckBAC();
204     1  FlagTxBAC = 2;
205     1  TimerTimeoutBAC = TimerRelogio3;
206     1  return;
207 }
208
209 void EnviaNackBAC (void)
210 {
211     1  TxMsgAckBAC();
212     1  FlagTxBAC = 2;
213     1  TimerTimeoutBAC = TimerRelogio3;
214     1  return;
215 }
216
217 void EnviaMsgBAC (BYTE *msg, BYTE n)
218 {
219     1  BYTE i;
220     1
221     1  /* [01] */
222     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
223         1  (n < MAXBFTXBAC))
224     1  {
225         2  /* Copia a mensagem para a fila. */
226         2  i = 0;
227         2  while (i < n)
228         2  {
229             3  m[i] = msg[i];
230             3  i++;
231         }
232         2  /* Sim. Copia a mensagem para o ponteiro. */
233         2  i = 0;
234         2  while (i < n)
235         2  {
236             3  m[i] = msg[i];
237             3  i++;
238         }
239         2  /* [02] */
240         2  /* Atualiza indices da fila de recepcao. */
241         2  disable();
242         2  if (PFRxBAC >= (MAXFRXBAC-1))
243         2  {
244             3  PFRxBAC = 0;
245         }
246         2  else
247         2  {
248             3  PFRxBAC ++;
249             3  QFRxBAC --;
250             3  enable();
251         }
252         2  return(0);
253     }
254     1  return(1);
255 }
256
257 void EnviaAckBAC (void)
258 {
259     1  TxMsgAckBAC();
260     1  FlagTxBAC = 2;
261     1  TimerTimeoutBAC = TimerRelogio3;
262     1  return;
263 }
264
265 void EnviaNackBAC (void)
266 {
267     1  TxMsgAckBAC();
268     1  FlagTxBAC = 2;
269     1  TimerTimeoutBAC = TimerRelogio3;
270     1  return;
271 }
272
273 void EnviaMsgBAC (BYTE *msg, BYTE n)
274 {
275     1  BYTE i;
276     1
277     1  /* [01] */
278     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
279         1  (n < MAXBFTXBAC))
280     1  {
281         2  /* Copia a mensagem para a fila. */
282         2  i = 0;
283         2  while (i < n)
284         2  {
285             3  m[i] = msg[i];
286             3  i++;
287         }
288         2  /* Sim. Copia a mensagem para o ponteiro. */
289         2  i = 0;
290         2  while (i < n)
291         2  {
292             3  m[i] = msg[i];
293             3  i++;
294         }
295         2  /* [02] */
296         2  /* Atualiza indices da fila de recepcao. */
297         2  disable();
298         2  if (PFRxBAC >= (MAXFRXBAC-1))
299         2  {
300             3  PFRxBAC = 0;
301         }
302         2  else
303         2  {
304             3  PFRxBAC ++;
305             3  QFRxBAC --;
306             3  enable();
307         }
308         2  return(0);
309     }
310     1  return(1);
311 }
312
313 void EnviaAckBAC (void)
314 {
315     1  TxMsgAckBAC();
316     1  FlagTxBAC = 2;
317     1  TimerTimeoutBAC = TimerRelogio3;
318     1  return;
319 }
320
321 void EnviaNackBAC (void)
322 {
323     1  TxMsgAckBAC();
324     1  FlagTxBAC = 2;
325     1  TimerTimeoutBAC = TimerRelogio3;
326     1  return;
327 }
328
329 void EnviaMsgBAC (BYTE *msg, BYTE n)
330 {
331     1  BYTE i;
332     1
333     1  /* [01] */
334     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
335         1  (n < MAXBFTXBAC))
336     1  {
337         2  /* Copia a mensagem para a fila. */
338         2  i = 0;
339         2  while (i < n)
340         2  {
341             3  m[i] = msg[i];
342             3  i++;
343         }
344         2  /* Sim. Copia a mensagem para o ponteiro. */
345         2  i = 0;
346         2  while (i < n)
347         2  {
348             3  m[i] = msg[i];
349             3  i++;
350         }
349         2  /* [02] */
350         2  /* Atualiza indices da fila de recepcao. */
351         2  disable();
352         2  if (PFRxBAC >= (MAXFRXBAC-1))
353         2  {
354             3  PFRxBAC = 0;
355         }
356         2  else
357         2  {
358             3  PFRxBAC ++;
359             3  QFRxBAC --;
360             3  enable();
361         }
362         2  return(0);
363     }
364     1  return(1);
365 }
366
367 void EnviaAckBAC (void)
368 {
369     1  TxMsgAckBAC();
370     1  FlagTxBAC = 2;
371     1  TimerTimeoutBAC = TimerRelogio3;
372     1  return;
373 }
374
375 void EnviaNackBAC (void)
376 {
377     1  TxMsgAckBAC();
378     1  FlagTxBAC = 2;
379     1  TimerTimeoutBAC = TimerRelogio3;
380     1  return;
381 }
382
383 void EnviaMsgBAC (BYTE *msg, BYTE n)
384 {
385     1  BYTE i;
386     1
387     1  /* [01] */
388     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
389         1  (n < MAXBFTXBAC))
390     1  {
391         2  /* Copia a mensagem para a fila. */
392         2  i = 0;
393         2  while (i < n)
394         2  {
395             3  m[i] = msg[i];
396             3  i++;
397         }
398         2  /* Sim. Copia a mensagem para o ponteiro. */
399         2  i = 0;
400         2  while (i < n)
401         2  {
402             3  m[i] = msg[i];
403             3  i++;
404         }
405         2  /* [02] */
406         2  /* Atualiza indices da fila de recepcao. */
407         2  disable();
408         2  if (PFRxBAC >= (MAXFRXBAC-1))
409         2  {
410             3  PFRxBAC = 0;
411         }
412         2  else
413         2  {
414             3  PFRxBAC ++;
415             3  QFRxBAC --;
416             3  enable();
417         }
418         2  return(0);
419     }
420     1  return(1);
421 }
422
423 void EnviaAckBAC (void)
424 {
425     1  TxMsgAckBAC();
426     1  FlagTxBAC = 2;
427     1  TimerTimeoutBAC = TimerRelogio3;
428     1  return;
429 }
430
431 void EnviaNackBAC (void)
432 {
433     1  TxMsgAckBAC();
434     1  FlagTxBAC = 2;
435     1  TimerTimeoutBAC = TimerRelogio3;
436     1  return;
437 }
438
439 void EnviaMsgBAC (BYTE *msg, BYTE n)
440 {
441     1  BYTE i;
442     1
443     1  /* [01] */
444     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
445         1  (n < MAXBFTXBAC))
446     1  {
447         2  /* Copia a mensagem para a fila. */
448         2  i = 0;
449         2  while (i < n)
450         2  {
451             3  m[i] = msg[i];
452             3  i++;
453         }
454         2  /* Sim. Copia a mensagem para o ponteiro. */
455         2  i = 0;
456         2  while (i < n)
457         2  {
458             3  m[i] = msg[i];
459             3  i++;
460         }
459         2  /* [02] */
460         2  /* Atualiza indices da fila de recepcao. */
461         2  disable();
462         2  if (PFRxBAC >= (MAXFRXBAC-1))
463         2  {
464             3  PFRxBAC = 0;
465         }
466         2  else
467         2  {
468             3  PFRxBAC ++;
469             3  QFRxBAC --;
470             3  enable();
471         }
472         2  return(0);
473     }
474     1  return(1);
475 }
476
477 void EnviaAckBAC (void)
478 {
479     1  TxMsgAckBAC();
480     1  FlagTxBAC = 2;
481     1  TimerTimeoutBAC = TimerRelogio3;
482     1  return;
483 }
484
485 void EnviaNackBAC (void)
486 {
487     1  TxMsgAckBAC();
488     1  FlagTxBAC = 2;
489     1  TimerTimeoutBAC = TimerRelogio3;
490     1  return;
491 }
492
493 void EnviaMsgBAC (BYTE *msg, BYTE n)
494 {
495     1  BYTE i;
496     1
497     1  /* [01] */
498     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
499         1  (n < MAXBFTXBAC))
500     1  {
501         2  /* Copia a mensagem para a fila. */
502         2  i = 0;
503         2  while (i < n)
504         2  {
505             3  m[i] = msg[i];
506             3  i++;
507         }
508         2  /* Sim. Copia a mensagem para o ponteiro. */
509         2  i = 0;
510         2  while (i < n)
511         2  {
512             3  m[i] = msg[i];
513             3  i++;
514         }
515         2  /* [02] */
516         2  /* Atualiza indices da fila de recepcao. */
517         2  disable();
518         2  if (PFRxBAC >= (MAXFRXBAC-1))
519         2  {
520             3  PFRxBAC = 0;
521         }
522         2  else
523         2  {
524             3  PFRxBAC ++;
525             3  QFRxBAC --;
526             3  enable();
527         }
528         2  return(0);
529     }
530     1  return(1);
531 }
532
533 void EnviaAckBAC (void)
534 {
535     1  TxMsgAckBAC();
536     1  FlagTxBAC = 2;
537     1  TimerTimeoutBAC = TimerRelogio3;
538     1  return;
539 }
540
541 void EnviaNackBAC (void)
542 {
543     1  TxMsgAckBAC();
544     1  FlagTxBAC = 2;
545     1  TimerTimeoutBAC = TimerRelogio3;
546     1  return;
547 }
548
549 void EnviaMsgBAC (BYTE *msg, BYTE n)
550 {
551     1  BYTE i;
552     1
553     1  /* [01] */
554     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
555         1  (n < MAXBFTXBAC))
556     1  {
557         2  /* Copia a mensagem para a fila. */
558         2  i = 0;
559         2  while (i < n)
560         2  {
561             3  m[i] = msg[i];
562             3  i++;
563         }
564         2  /* Sim. Copia a mensagem para o ponteiro. */
565         2  i = 0;
566         2  while (i < n)
567         2  {
568             3  m[i] = msg[i];
569             3  i++;
570         }
569         2  /* [02] */
570         2  /* Atualiza indices da fila de recepcao. */
571         2  disable();
572         2  if (PFRxBAC >= (MAXFRXBAC-1))
573         2  {
574             3  PFRxBAC = 0;
575         }
576         2  else
577         2  {
578             3  PFRxBAC ++;
579             3  QFRxBAC --;
580             3  enable();
581         }
582         2  return(0);
583     }
584     1  return(1);
585 }
586
587 void EnviaAckBAC (void)
588 {
589     1  TxMsgAckBAC();
590     1  FlagTxBAC = 2;
591     1  TimerTimeoutBAC = TimerRelogio3;
592     1  return;
593 }
594
595 void EnviaNackBAC (void)
596 {
597     1  TxMsgAckBAC();
598     1  FlagTxBAC = 2;
599     1  TimerTimeoutBAC = TimerRelogio3;
600     1  return;
601 }
602
603 void EnviaMsgBAC (BYTE *msg, BYTE n)
604 {
605     1  BYTE i;
606     1
607     1  /* [01] */
608     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
609         1  (n < MAXBFTXBAC))
610     1  {
611         2  /* Copia a mensagem para a fila. */
612         2  i = 0;
613         2  while (i < n)
614         2  {
615             3  m[i] = msg[i];
616             3  i++;
617         }
618         2  /* Sim. Copia a mensagem para o ponteiro. */
619         2  i = 0;
620         2  while (i < n)
621         2  {
622             3  m[i] = msg[i];
623             3  i++;
624         }
625         2  /* [02] */
626         2  /* Atualiza indices da fila de recepcao. */
627         2  disable();
628         2  if (PFRxBAC >= (MAXFRXBAC-1))
629         2  {
630             3  PFRxBAC = 0;
631         }
632         2  else
633         2  {
634             3  PFRxBAC ++;
635             3  QFRxBAC --;
636             3  enable();
637         }
638         2  return(0);
639     }
640     1  return(1);
641 }
642
643 void EnviaAckBAC (void)
644 {
645     1  TxMsgAckBAC();
646     1  FlagTxBAC = 2;
647     1  TimerTimeoutBAC = TimerRelogio3;
648     1  return;
649 }
650
651 void EnviaNackBAC (void)
652 {
653     1  TxMsgAckBAC();
654     1  FlagTxBAC = 2;
655     1  TimerTimeoutBAC = TimerRelogio3;
656     1  return;
657 }
658
659 void EnviaMsgBAC (BYTE *msg, BYTE n)
660 {
661     1  BYTE i;
662     1
663     1  /* [01] */
664     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
665         1  (n < MAXBFTXBAC))
666     1  {
667         2  /* Copia a mensagem para a fila. */
668         2  i = 0;
669         2  while (i < n)
670         2  {
671             3  m[i] = msg[i];
672             3  i++;
673         }
674         2  /* Sim. Copia a mensagem para o ponteiro. */
675         2  i = 0;
676         2  while (i < n)
677         2  {
678             3  m[i] = msg[i];
679             3  i++;
680         }
679         2  /* [02] */
680         2  /* Atualiza indices da fila de recepcao. */
681         2  disable();
682         2  if (PFRxBAC >= (MAXFRXBAC-1))
683         2  {
684             3  PFRxBAC = 0;
685         }
686         2  else
687         2  {
688             3  PFRxBAC ++;
689             3  QFRxBAC --;
690             3  enable();
691         }
692         2  return(0);
693     }
694     1  return(1);
695 }
696
697 void EnviaAckBAC (void)
698 {
699     1  TxMsgAckBAC();
700     1  FlagTxBAC = 2;
701     1  TimerTimeoutBAC = TimerRelogio3;
702     1  return;
703 }
704
705 void EnviaNackBAC (void)
706 {
707     1  TxMsgAckBAC();
708     1  FlagTxBAC = 2;
709     1  TimerTimeoutBAC = TimerRelogio3;
710     1  return;
711 }
712
713 void EnviaMsgBAC (BYTE *msg, BYTE n)
714 {
715     1  BYTE i;
716     1
717     1  /* [01] */
718     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
719         1  (n < MAXBFTXBAC))
720     1  {
721         2  /* Copia a mensagem para a fila. */
722         2  i = 0;
723         2  while (i < n)
724         2  {
725             3  m[i] = msg[i];
726             3  i++;
727         }
728         2  /* Sim. Copia a mensagem para o ponteiro. */
729         2  i = 0;
730         2  while (i < n)
731         2  {
732             3  m[i] = msg[i];
733             3  i++;
734         }
733         2  /* [02] */
734         2  /* Atualiza indices da fila de recepcao. */
735         2  disable();
736         2  if (PFRxBAC >= (MAXFRXBAC-1))
737         2  {
738             3  PFRxBAC = 0;
739         }
740         2  else
741         2  {
742             3  PFRxBAC ++;
743             3  QFRxBAC --;
744             3  enable();
745         }
746         2  return(0);
747     }
748     1  return(1);
749 }
750
751 void EnviaAckBAC (void)
752 {
753     1  TxMsgAckBAC();
754     1  FlagTxBAC = 2;
755     1  TimerTimeoutBAC = TimerRelogio3;
756     1  return;
757 }
758
759 void EnviaNackBAC (void)
760 {
761     1  TxMsgAckBAC();
762     1  FlagTxBAC = 2;
763     1  TimerTimeoutBAC = TimerRelogio3;
764     1  return;
765 }
766
767 void EnviaMsgBAC (BYTE *msg, BYTE n)
768 {
769     1  BYTE i;
770     1
771     1  /* [01] */
772     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
773         1  (n < MAXBFTXBAC))
774     1  {
775         2  /* Copia a mensagem para a fila. */
776         2  i = 0;
777         2  while (i < n)
778         2  {
779             3  m[i] = msg[i];
780             3  i++;
781         }
782         2  /* Sim. Copia a mensagem para o ponteiro. */
783         2  i = 0;
784         2  while (i < n)
785         2  {
786             3  m[i] = msg[i];
787             3  i++;
788         }
787         2  /* [02] */
788         2  /* Atualiza indices da fila de recepcao. */
789         2  disable();
790         2  if (PFRxBAC >= (MAXFRXBAC-1))
791         2  {
792             3  PFRxBAC = 0;
793         }
794         2  else
795         2  {
796             3  PFRxBAC ++;
797             3  QFRxBAC --;
798             3  enable();
799         }
798         2  return(0);
799     }
800     1  return(1);
801 }
802
803 void EnviaAckBAC (void)
804 {
805     1  TxMsgAckBAC();
806     1  FlagTxBAC = 2;
807     1  TimerTimeoutBAC = TimerRelogio3;
808     1  return;
809 }
810
811 void EnviaNackBAC (void)
812 {
813     1  TxMsgAckBAC();
814     1  FlagTxBAC = 2;
815     1  TimerTimeoutBAC = TimerRelogio3;
816     1  return;
817 }
818
819 void EnviaMsgBAC (BYTE *msg, BYTE n)
820 {
821     1  BYTE i;
822     1
823     1  /* [01] */
824     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
825         1  (n < MAXBFTXBAC))
826     1  {
827         2  /* Copia a mensagem para a fila. */
828         2  i = 0;
829         2  while (i < n)
830         2  {
831             3  m[i] = msg[i];
832             3  i++;
833         }
834         2  /* Sim. Copia a mensagem para o ponteiro. */
835         2  i = 0;
836         2  while (i < n)
837         2  {
838             3  m[i] = msg[i];
839             3  i++;
840         }
839         2  /* [02] */
840         2  /* Atualiza indices da fila de recepcao. */
841         2  disable();
842         2  if (PFRxBAC >= (MAXFRXBAC-1))
843         2  {
844             3  PFRxBAC = 0;
845         }
846         2  else
847         2  {
848             3  PFRxBAC ++;
849             3  QFRxBAC --;
850             3  enable();
851         }
848         2  return(0);
849     }
850     1  return(1);
851 }
852
853 void EnviaAckBAC (void)
854 {
855     1  TxMsgAckBAC();
856     1  FlagTxBAC = 2;
857     1  TimerTimeoutBAC = TimerRelogio3;
858     1  return;
859 }
860
861 void EnviaNackBAC (void)
862 {
863     1  TxMsgAckBAC();
864     1  FlagTxBAC = 2;
865     1  TimerTimeoutBAC = TimerRelogio3;
866     1  return;
867 }
868
869 void EnviaMsgBAC (BYTE *msg, BYTE n)
870 {
871     1  BYTE i;
872     1
873     1  /* [01] */
874     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
875         1  (n < MAXBFTXBAC))
876     1  {
877         2  /* Copia a mensagem para a fila. */
878         2  i = 0;
879         2  while (i < n)
880         2  {
881             3  m[i] = msg[i];
882             3  i++;
883         }
884         2  /* Sim. Copia a mensagem para o ponteiro. */
885         2  i = 0;
886         2  while (i < n)
887         2  {
888             3  m[i] = msg[i];
889             3  i++;
890         }
889         2  /* [02] */
890         2  /* Atualiza indices da fila de recepcao. */
891         2  disable();
892         2  if (PFRxBAC >= (MAXFRXBAC-1))
893         2  {
894             3  PFRxBAC = 0;
895         }
896         2  else
897         2  {
898             3  PFRxBAC ++;
899             3  QFRxBAC --;
900             3  enable();
901         }
900         2  return(0);
901     }
902     1  return(1);
903 }
904
905 void EnviaAckBAC (void)
906 {
907     1  TxMsgAckBAC();
908     1  FlagTxBAC = 2;
909     1  TimerTimeoutBAC = TimerRelogio3;
910     1  return;
911 }
912
913 void EnviaNackBAC (void)
914 {
915     1  TxMsgAckBAC();
916     1  FlagTxBAC = 2;
917     1  TimerTimeoutBAC = TimerRelogio3;
918     1  return;
919 }
920
921 void EnviaMsgBAC (BYTE *msg, BYTE n)
922 {
923     1  BYTE i;
924     1
925     1  /* [01] */
926     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
927         1  (n < MAXBFTXBAC))
928     1  {
929         2  /* Copia a mensagem para a fila. */
930         2  i = 0;
931         2  while (i < n)
932         2  {
933             3  m[i] = msg[i];
934             3  i++;
935         }
936         2  /* Sim. Copia a mensagem para o ponteiro. */
937         2  i = 0;
938         2  while (i < n)
939         2  {
940             3  m[i] = msg[i];
941             3  i++;
942         }
941         2  /* [02] */
942         2  /* Atualiza indices da fila de recepcao. */
943         2  disable();
944         2  if (PFRxBAC >= (MAXFRXBAC-1))
945         2  {
946             3  PFRxBAC = 0;
947         }
948         2  else
949         2  {
950             3  PFRxBAC ++;
951             3  QFRxBAC --;
952             3  enable();
953         }
952         2  return(0);
953     }
954     1  return(1);
955 }
956
957 void EnviaAckBAC (void)
958 {
959     1  TxMsgAckBAC();
960     1  FlagTxBAC = 2;
961     1  TimerTimeoutBAC = TimerRelogio3;
962     1  return;
963 }
964
965 void EnviaNackBAC (void)
966 {
967     1  TxMsgAckBAC();
968     1  FlagTxBAC = 2;
969     1  TimerTimeoutBAC = TimerRelogio3;
970     1  return;
971 }
972
973 void EnviaMsgBAC (BYTE *msg, BYTE n)
974 {
975     1  BYTE i;
976     1
977     1  /* [01] */
978     1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
979         1  (n < MAXBFTXBAC))
980     1  {
981         2  /* Copia a mensagem para a fila. */
982         2  i = 0;
983         2  while (i < n)
984         2  {
985             3  m[i] = msg[i];
986             3  i++;
987         }
988         2  /* Sim. Copia a mensagem para o ponteiro. */
989         2  i = 0;
990         2  while (i < n)
991         2  {
992             3  m[i] = msg[i];
993             3  i++;
994         }
993         2  /* [02] */
994         2  /* Atualiza indices da fila de recepcao. */
995         2  disable();
996         2  if (PFRxBAC >= (MAXFRXBAC-1))
997         2  {
998             3  PFRxBAC = 0;
999         }
1000        2  else
1001        2  {
1002            3  PFRxBAC ++;
1003            3  QFRxBAC --;
1004            3  enable();
1005        }
1006        2  return(0);
1007    }
1008    1  return(1);
1009 }
1010
1011 void EnviaAckBAC (void)
1012 {
1013    1  TxMsgAckBAC();
1014    1  FlagTxBAC = 2;
1015    1  TimerTimeoutBAC = TimerRelogio3;
1016    1  return;
1017 }
1018
1019 void EnviaNackBAC (void)
1020 {
1021    1  TxMsgAckBAC();
1022    1  FlagTxBAC = 2;
1023    1  TimerTimeoutBAC = TimerRelogio3;
1024    1  return;
1025 }
1026
1027 void EnviaMsgBAC (BYTE *msg, BYTE n)
1028 {
1029    1  BYTE i;
1030    1
1031    1  /* [01] */
1032    1  if ((SemaforoTxBAC == 0) && (QFRxBAC < MAXFTXBAC) &&
1033        1  (n < MAXBFTXBAC))
1034    1  {
1035        2  /* Copia a mensagem para a fila. */
1036        2  i = 0;
1037        2  while (i < n)
1038        2  {
1039            3  m
```

Qualidade & Segurança - Definições

Qualidade - indicadores

Confiabilidade

Funcionalidade

Facilidade de uso

Desempenho

Capacidade

Documentação

Suporte, Instalação e

Manutenção

Segurança - $S(t)$

Probabilidade de um sistema
efetuar suas operações de
forma correta ou
descontinuar seu
funcionamento de forma a
não comprometer a operação
de outros sistemas ou
comprometer a Segurança

Qualidade & Segurança - Definições

Qual

S(t)

sistema

ações de

a ou

seu

forma a

operação

as ou

gurança



Medir a Confiabilidade:

Observação de falhas

Análise estatística.

Problemas:

- Início na etapa de testes:

Produto deve estar terminado

- Tempo necessário:

Software Ultra-confiabilidade

- Informações ignoradas:

Métodos empregados

Alternativa:

```
void IniciaBAC (void)
```

Medição da Confiabilidade

Medir a Confiabilidade:

Observação de falhas

Análise estatística.

Problemas:

- Início na etapa de testes:

- Produto deve estar terminado

- Tempo necessário:

- Software Ultra-confiabilidade

- Informações ignoradas:

- Métodos empregados

Alternativa:

Especificação e Análise dos Processos de Desenvolvimento de Software.

Conformidade com Normas Técnicas adequadas e Modelos de ampla aceitação.

Avaliação da Confiabilidade sobre Artefatos Intermediários de cada fase do Processo de Desenvolvimento.

Representação em estágios:

A Capacidade do Processo
estabelece o seu
grau de **refinamento** e
institucionalização.

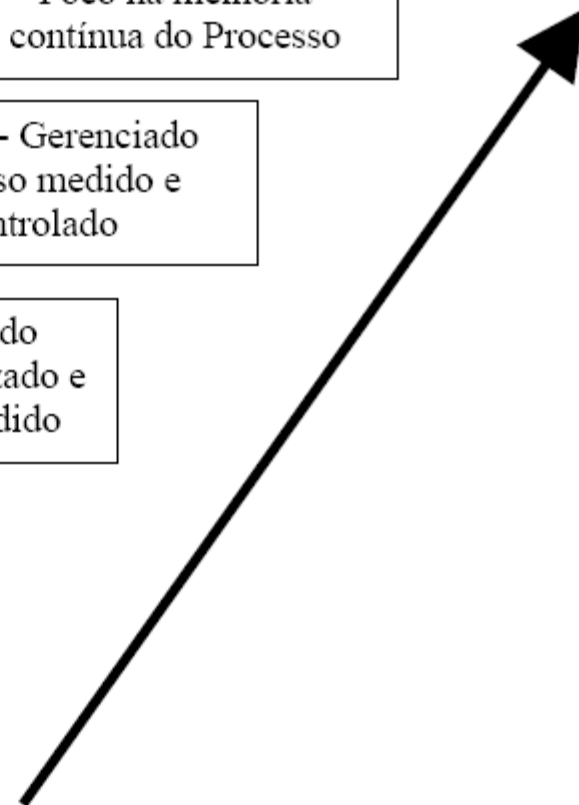
Nível 1 - Inicial
Fracamente Controlado,
Imprevisível

Nível 2 - Repetível
Repetição de Tarefas
executadas com sucesso

Nível 3 - Definido
Processo caracterizado e
claramente entendido

Nível 4 - Gerenciado
Processo medido e
controlado

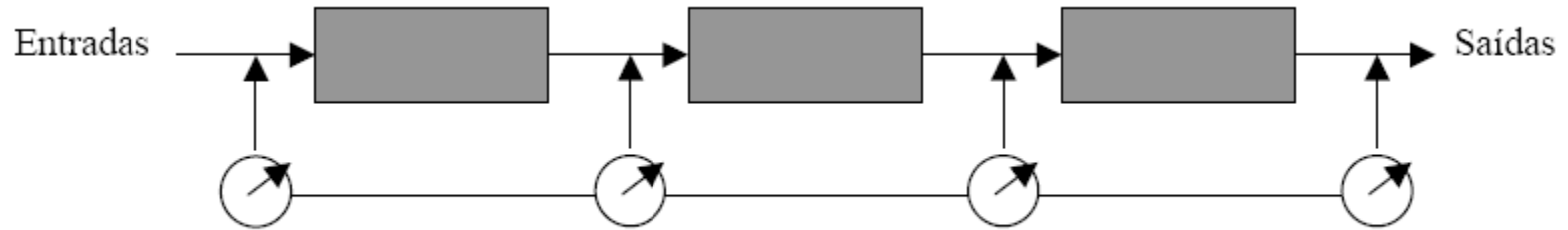
Nível 5 - Otimização
Foco na melhoria
contínua do Processo



CMU / SEI - CMMi

Visibilidade:

CMMi nível 2



```

89 void IniciaBAC (void)
90 {
91     Diagnostico[0x07] ++;
92     UFtxBAC = 0;
93     PFtxBAC = 0;
94     AFtxBAC = 0;
95     QFtxBAC = 0;
96     UFRxBAC = 0;
  
```

```

202 3 FtxBAC[UFtxBAC].m[1+1] = 0;
203 3 }
204 2
205 2 FtxBAC[UFtxBAC].m[1+1] = 0;
206 2
207 2
208 2
209 2 /* Atualiza o status da fila de transmissao. */
  
```

```

197 1 {
198 2 /* Copia a mensagem para a fila. */
199 2 i = 0;
200 2 while (i < n)
  
```

```

305 1 FlagtxBAC = 2;
306 1 TimerTimeoutBAC = TimerRelogio3;
307 1 return;
308 1 }
333 void EnviaNackBAC (void)
  
```

Requisito Técnico de Software - Qualidade

Deve ser adotada a metodologia estabelecida no “Capability Maturity Model Integration for Systems Engineering and Software Engineering” (CMMI-SE/SW), versão 1.1 ou superior, do “Software Engineering Institute / Carnegie Mellon University” (SEI/CMU) para todo o Software fornecido, sendo que:

- Deve ser adotado **CMMI-SE/SW nível dois** (2) ou superior, na representação em estágios “Staged Representation”;
- Deve ser comprovado pelo “Standard CMMI Appraisal Method for Process Improvement” (**SCAMPI**), o emprego desta metodologia por **toda equipe** envolvida no projeto.

[EPS 2004]

Fail Safe (Falha Segura)

A validade deste conceito é, em geral, baseada na experiência, mas tem **aplicação limitada** ao desenvolvimento e uso de sistemas grandes e complexos empregando microprocessadores comerciais.

O crescimento exponencial no número de combinações de falhas a serem consideradas quando usando tais componentes significam que uma *abordagem determinística* é, geralmente impraticável. Com tais sistemas complexos, a **abordagem probabilística** pode ser usada efetivamente.

[CENELEC 50126]

Exemplos de aplicação em Software:

- Rotinas de diagnóstico
- Programação Defensiva

```
void IniciaBAC (void)
```

```
{
```

```
  Diagnostico[0x07]
```

```
  UFTxBAC = 0;
```

```
  PFTxBAC = 0;
```

```
  AFTxBAC = 0;
```

```
  QFTxBAC = 0;
```

```
  UFRxBAC = 0;
```

```
  PFRxBAC = 0;
```

```
  QFRxBAC = 0;
```

```
}
```

Níveis de Risco

Determinação dos Níveis de Risco

* Freqüência da ocorrência de um evento de ameaça	Níveis de Risco			
Freqüente	Indesejável	Intolerável	Intolerável	Intolerável
Provável	Tolerável	Indesejável	Intolerável	Intolerável
Ocasional	Tolerável	Indesejável	Indesejável	Intolerável
Remoto	Negligenciável	Tolerável	Indesejável	Indesejável
Improvável	Negligenciável	Negligenciável	Tolerável	Tolerável
Inacreditável	Negligenciável	Negligenciável	Negligenciável	Negligenciável
	Insignificante	Marginal	Crítico	Catastrófico
	Níveis de gravidade da consequência da ameaça.			

```

89 void IniciaBAC (void)
90 {
91     Diagnostico[0x07] ++;
92     UFTxBAC = 0;
93     PFTxBAC = 0;
94     AFTxBAC = 0;
95     QFTxBAC = 0;
96     UFRxBAC = 0;
97     PFRxBAC = 0;
98     QFRxBAC = 0;
99     FlagEstadoBAC = 0;
100     FlagCncBAC = 0;

```

```

202     3     FTxBAC[UFTxBAC] = 0;
203     3     }
204     2     FTxBAC[UFTxBAC] = 0;
205     2     }
206     2     /* Atualiza o valor da fila de transmissao. */
207     2     /* (n = FTxBAC-1) */
208     2     }
209     2     }
210     2     }
211     2     }
212     2     }
213     2     }

```

```

196     1     (n < MAXBFTXBAC))
197     1     {
198     2     /* Copia a mensagem para a fila. */
199     2     i = 0;
200     2     while (i < n)
201     2     {

```

```

305     1     FlagTxBAC = 2;
306     1     TimerTimeoutBAC = TimerRelogio3;
307     1     return;
308     1     }
309     1     void EnviaNackBAC (void)

```

IEC 61508 - Parâmetros para determinar SIL

Consequência do Risco

- C1 Perdas Pequenas.
- C2 Perdas sérias e permanentes para uma ou mais pessoas, ou morte de uma pessoa.
- C3 Morte de várias pessoas.
- C4 Muitas pessoas assassinadas.

Frequência e Tempo de Exposição

- F1 Exposição de rara a freqüente na região de perigo.
- F2 Exposição de freqüente a permanente na região de perigo.

Possibilidade de evitar o evento

- P1 Possível sob certas condições.
- P2 Quase impossível.

Probabilidade de ocorrências indesejáveis

- W1 Probabilidade muito pequena de que as ocorrências aconteçam.
- W2 Probabilidade pequena de que as ocorrências aconteçam.
- W3 Probabilidade relativamente alta de que as ocorrências aconteçam.

```

89 void IniciaBAC (void)
90 {
91     UFTxBAC = 0;
92     PFTxBAC = 0;
93     AFTxBAC = 0;
94     QFTxBAC = 0;
95     UFRxBAC = 0;
96     PFRxBAC = 0;
97     QFRxBAC = 0;
98     FlagTxBAC = 0;
99     FlagEstadoBAC = 0;
100     FlagCpcBAC = 0;
101     FalhasBAC = 0;
102     SemaforoTxBAC = 0;
103     TimerTimeoutBAC = 0;
104     TimerRelogio3 = 0;
105     ProgramaRS232C();
106     while (1)
107     {
108         #if MESTREBAC == 1
109             return;
110         #endif
111         if ((PFRxBAC >= MAXFRxBAC) || (UFRxBAC >= MAXURxBAC) || (QFRxBAC >= MAXQRxBAC))
112         {
113             disable();
114             if ((PFRxBAC >= MAXFRxBAC) || (UFRxBAC >= MAXURxBAC) || (QFRxBAC >= MAXQRxBAC))
115             {
116                 BYTE i;
117                 while (i < n)
118                 {
119                     /* Copia a mensagem para a fila. */
120                     i = 0;
121                     while (i < n)
122                     {
123                         BYTE j;
124                         while (j < n)
125                         {
126                             BYTE k;
127                             while (k < n)
128                             {
129                                 BYTE l;
130                                 while (l < n)
131                                 {
132                                     BYTE m;
133                                     while (m < n)
134                                     {
135                                         BYTE o;
136                                         while (o < n)
137                                         {
138                                             BYTE p;
139                                             while (p < n)
140                                             {
141                                                 BYTE q;
142                                                 while (q < n)
143                                                 {
144                                                     BYTE r;
145                                                     while (r < n)
146                                                     {
147                                                         BYTE s;
148                                                         while (s < n)
149                                                         {
150                                                             BYTE t;
151                                                             while (t < n)
152                                                             {
153                                                                 BYTE u;
154                                                                 while (u < n)
155                                                                 {
156                                                                     BYTE v;
157                                                                     while (v < n)
158                                                                     {
159                                                                         BYTE w;
160                                                                         while (w < n)
161                                                                         {
162                                                                             BYTE x;
163                                                                             while (x < n)
164                                                                             {
165                                                                                 BYTE y;
166                                                                                 while (y < n)
167                                                                                 {
168                                                                                     BYTE z;
169                                                                                     while (z < n)
170                                                                                     {
171                                                                                         BYTE aa;
172                                                                                         while (aa < n)
173                                                                                         {
174                                                                                             BYTE ab;
175                                                                                             while (ab < n)
176                                                                                             {
177                                                                                                 BYTE ac;
178                                                                                                 while (ac < n)
179                                                                                                 {
180                                                                                                     BYTE ad;
181                                                                                                     while (ad < n)
182                                                                                                     {
183                                                                                                         BYTE ae;
184                                                                                                         while (ae < n)
185                                                                                                         {
186                                                                                                             BYTE af;
187                                                                                                             while (af < n)
188                                                                                                             {
189                                                                                                                 BYTE ag;
190                                                                                                                 while (ag < n)
191                                                                                                                 {
192                                                                                                                     BYTE ah;
193                                                                                                                     while (ah < n)
194                                                                                                                     {
195                                                                                                                         BYTE ai;
196                                                                                                                         while (ai < n)
197                                                                                                                         {
198                                                                                                                             BYTE aj;
199                                                                                                                             while (aj < n)
200                                                                                                                             {
201                                                                                                                         BYTE ak;
202                                                                                                                         while (ak < n)
203                                                                                                                         {
204                                                                                                                             BYTE al;
205                                                                                                                             while (al < n)
206                                                                                                                             {
207                                                                                                                         BYTE am;
208                                                                                                                         while (am < n)
209                                                                                                                         {
210                                                                                                                             BYTE an;
211                                                                                                                             while (an < n)
212                                                                                                                             {
213                                                                                                                         BYTE ao;
214                                                                                                                         while (ao < n)
215                                                                                                                         {
216                                                                                                                             BYTE ap;
217                                                                                                                             while (ap < n)
218                                                                                                                             {
219                                                                                                                         BYTE aq;
220                                                                                                                         while (aq < n)
221                                                                                                                         {
222                                                                                                                             BYTE ar;
223                                                                                                                             while (ar < n)
224                                                                                                                             {
225                                                                                                                         BYTE as;
226                                                                                                                         while (as < n)
227                                                                                                                         {
228                                                                                                                             BYTE at;
229                                                                                                                             while (at < n)
230                                                                                                                             {
231                                                                                                                         BYTE au;
232                                                                                                                         while (au < n)
233                                                                                                                         {
234                                                                                                                             BYTE av;
235                                                                                                                             while (av < n)
236                                                                                                                             {
237                                                                                                                         BYTE aw;
238                                                                                                                         while (aw < n)
239                                                                                                                         {
240                                                                                                                             BYTE ax;
241                                                                                                                             while (ax < n)
242                                                                                                                             {
243                                                                                                                         BYTE ay;
244                                                                                                                         while (ay < n)
245                                                                                                                         {
246                                                                                                                             BYTE az;
247                                                                                                                             while (az < n)
248                                                                                                                             {
249                                                                                                                         BYTE ba;
250                                                                                                                         while (ba < n)
251                                                                                                                         {
252                                                                                                                             BYTE bb;
253                                                                                                                             while (bb < n)
254                                                                                                                             {
255                                                                                                                         BYTE bc;
256                                                                                                                         while (bc < n)
257                                                                                                                         {
258                                                                                                                             BYTE bd;
259                                                                                                                             while (bd < n)
260                                                                                                                             {
261                                                                                                                         BYTE be;
262                                                                                                                         while (be < n)
263                                                                                                                         {
264                                                                                                                             BYTE bf;
265                                                                                                                             while (bf < n)
266                                                                                                                             {
267                                                                                                                         BYTE bg;
268                                                                                                                         while (bg < n)
269                                                                                                                         {
270                                                                                                                             BYTE bh;
271                                                                                                                             while (bh < n)
272                                                                                                                             {
273                                                                                                                         BYTE bi;
274                                                                                                                         while (bi < n)
275                                                                                                                         {
276                                                                                                                         BYTE bj;
277                                                                                                                         while (bj < n)
278                                                                                                                         {
279                                                                                                                         BYTE bk;
280                                                                                                                         while (bk < n)
281                                                                                                                         {
282                                                                                                                         BYTE bl;
283                                                                                                                         while (bl < n)
284                                                                                                                         {
285                                                                                                                         BYTE bm;
286                                                                                                                         while (bm < n)
287                                                                                                                         {
288                                                                                                                         BYTE bn;
289                                                                                                                         while (bn < n)
290                                                                                                                         {
291                                                                                                                         BYTE bo;
292                                                                                                                         while (bo < n)
293                                                                                                                         {
294                                                                                                                         BYTE bp;
295                                                                                                                         while (bp < n)
296                                                                                                                         {
297                                                                                                                         BYTE bq;
298                                                                                                                         while (bq < n)
299                                                                                                                         {
300                                                                                                                         BYTE br;
301                                                                                                                         while (br < n)
302                                                                                                                         {
303                                                                                                                         BYTE bs;
304                                                                                                                         while (bs < n)
305                                                                                                                         {
306                                                                                                                         BYTE bt;
307                                                                                                                         while (bt < n)
308                                                                                                                         {
309                                                                                                                         BYTE bu;
310                                                                                                                         while (bu < n)
311                                                                                                                         {
312                                                                                                                         BYTE bv;
313                                                                                                                         while (bv < n)
314                                                                                                                         {
315                                                                                                                         BYTE bw;
316                                                                                                                         while (bw < n)
317                                                                                                                         {
318                                                                                                                         BYTE bx;
319                                                                                                                         while (bx < n)
320                                                                                                                         {
321                                                                                                                         BYTE by;
322                                                                                                                         while (by < n)
323                                                                                                                         {
324                                                                                                                         BYTE bz;
325                                                                                                                         while (bz < n)
326                                                                                                                         {
327                                                                                                                         BYTE ca;
328                                                                                                                         while (ca < n)
329                                                                                                                         {
330                                                                                                                         BYTE cb;
331                                                                                                                         while (cb < n)
332                                                                                                                         {
333                                                                                                                         BYTE cc;
334                                                                                                                         while (cc < n)
335                                                                                                                         {
336                                                                                                                         BYTE cd;
337                                                                                                                         while (cd < n)
338                                                                                                                         {
339                                                                                                                         BYTE ce;
340                                                                                                                         while (ce < n)
341                                                                                                                         {
342                                                                                                                         BYTE cf;
343                                                                                                                         while (cf < n)
344                                                                                                                         {
345                                                                                                                         BYTE cg;
346                                                                                                                         while (cg < n)
347                                                                                                                         {
348                                                                                                                         BYTE ch;
349                                                                                                                         while (ch < n)
350                                                                                                                         {
351                                                                                                                         BYTE ci;
352                                                                                                                         while (ci < n)
353                                                                                                                         {
354                                                                                                                         BYTE cj;
355                                                                                                                         while (cj < n)
356                                                                                                                         {
357                                                                                                                         BYTE ck;
358                                                                                                                         while (ck < n)
359                                                                                                                         {
360                                                                                                                         BYTE cl;
361                                                                                                                         while (cl < n)
362                                                                                                                         {
363                                                                                                                         BYTE cm;
364                                                                                                                         while (cm < n)
365                                                                                                                         {
366                                                                                                                         BYTE cn;
367                                                                                                                         while (cn < n)
368                                                                                                                         {
369                                                                                                                         BYTE co;
370                                                                                                                         while (co < n)
371                                                                                                                         {
372                                                                                                                         BYTE cp;
373                                                                                                                         while (cp < n)
374                                                                                                                         {
375                                                                                                                         BYTE cq;
376                                                                                                                         while (cq < n)
377                                                                                                                         {
378                                                                                                                         BYTE cr;
379                                                                                                                         while (cr < n)
380                                                                                                                         {
381                                                                                                                         BYTE cs;
382                                                                                                                         while (cs < n)
383                                                                                                                         {
384                                                                                                                         BYTE ct;
385                                                                                                                         while (ct < n)
386                                                                                                                         {
387                                                                                                                         BYTE cu;
388                                                                                                                         while (cu < n)
389                                                                                                                         {
390                                                                                                                         BYTE cv;
391                                                                                                                         while (cv < n)
392                                                                                                                         {
393                                                                                                                         BYTE cw;
394                                                                                                                         while (cw < n)
395                                                                                                                         {
396                                                                                                                         BYTE cx;
397                                                                                                                         while (cx < n)
398                                                                                                                         {
399                                                                                                                         BYTE cy;
400                                                                                                                         while (cy < n)
401                                                                                                                         {
402                                                                                                                         BYTE cz;
403                                                                                                                         while (cz < n)
404                                                                                                                         {
405                                                                                                                         BYTE da;
406                                                                                                                         while (da < n)
407                                                                                                                         {
408                                                                                                                         BYTE db;
409                                                                                                                         while (db < n)
410                                                                                                                         {
411                                                                                                                         BYTE dc;
412                                                                                                                         while (dc < n)
413                                                                                                                         {
414                                                                                                                         BYTE dd;
415                                                                                                                         while (dd < n)
416                                                                                                                         {
417                                                                                                                         BYTE de;
418                                                                                                                         while (de < n)
419                                                                                                                         {
420                                                                                                                         BYTE df;
421                                                                                                                         while (df < n)
422                                                                                                                         {
423                                                                                                                         BYTE dg;
424                                                                                                                         while (dg < n)
425                                                                                                                         {
426                                                                                                                         BYTE dh;
427                                                                                                                         while (dh < n)
428                                                                                                                         {
429                                                                                                                         BYTE di;
430                                                                                                                         while (di < n)
431                                                                                                                         {
432                                                                                                                         BYTE dj;
433                                                                                                                         while (dj < n)
434                                                                                                                         {
435                                                                                                                         BYTE dk;
436                                                                                                                         while (dk < n)
437                                                                                                                         {
438                                                                                                                         BYTE dl;
439                                                                                                                         while (dl < n)
440                                                                                                                         {
441                                                                                                                         BYTE dm;
442                                                                                                                         while (dm < n)
443                                                                                                                         {
444                                                                                                                         BYTE dn;
445                                                                                                                         while (dn < n)
446                                                                                                                         {
447                                                                                                                         BYTE do;
448                                                                                                                         while (do < n)
449                                                                                                                         {
450                                                                                                                         BYTE dp;
451                                                                                                                         while (dp < n)
452                                                                                                                         {
453                                                                                                                         BYTE dq;
454                                                                                                                         while (dq < n)
455                                                                                                                         {
456                                                                                                                         BYTE dr;
457                                                                                                                         while (dr < n)
458                                                                                                                         {
459                                                                                                                         BYTE ds;
460                                                                                                                         while (ds < n)
461                                                                                                                         {
462                                                                                                                         BYTE dt;
463                                                                                                                         while (dt < n)
464                                                                                                                         {
465                                                                                                                         BYTE du;
466                                                                                                                         while (du < n)
467                                                                                                                         {
468                                                                                                                         BYTE dv;
469                                                                                                                         while (dv < n)
470                                                                                                                         {
471                                                                                                                         BYTE dw;
472                                                                                                                         while (dw < n)
473                                                                                                                         {
474                                                                                                                         BYTE dx;
475                                                                                                                         while (dx < n)
476                                                                                                                         {
477                                                                                                                         BYTE dy;
478                                                                                                                         while (dy < n)
479                                                                                                                         {
480                                                                                                                         BYTE dz;
481                                                                                                                         while (dz < n)
482                                                                                                                         {
483                                                                                                                         BYTE ea;
484                                                                                                                         while (ea < n)
485                                                                                                                         {
486                                                                                                                         BYTE eb;
487                                                                                                                         while (eb < n)
488                                                                                                                         {
489                                                                                                                         BYTE ec;
490                                                                                                                         while (ec < n)
491                                                                                                                         {
492                                                                                                                         BYTE ed;
493                                                                                                                         while (ed < n)
494                                                                                                                         {
495                                                                                                                         BYTE ee;
496                                                                                                                         while (ee < n)
497                                                                                                                         {
498                                                                                                                         BYTE ef;
499                                                                                                                         while (ef < n)
500                                                                                                                         {
499

```

IEC 61508 - Parâmetros para determinar SIL

Consequência do Risco

- C1 Perdas Pequenas.
- C2 Perdas sérias e permanentes para uma ou mais pessoas, ou morte de uma pessoa.
- C3 Morte de várias pessoas.
- C4 Muitas pessoas assassinadas.

Frequência e Tempo de Exposição

- F1 Exposição de rara a freqüente na região de perigo.
- F2 Exposição de freqüente a permanente na região de perigo.

Possibilidade de evitar o evento

Probabilidade de ocorrências indesejáveis

- ```

1 g[i] = PFrxBAC[PFrxBAC].m[i+1];
2 }
2
2 /10024/
2 /* Atualiza indices da fila de rececao. */
2 g[i] = 0;
2 if (PFrxBAC >= (MAXFRXBAC-1))
2 PFrxBAC = 0;
2 else
2 PFrxBAC ++;
2 gFrxBAC = PFrxBAC;
2 enable();
2
2 // retorna
2 }
1
1 return 0;

```



# IEC 61508 - Parâmetros para determinar SIL

## Consequência do Risco

- C1 Perdas Pequenas.
- C2 Perdas sérias e permanentes para uma ou mais pessoas, ou morte de uma pessoa.
- C3 Morte de várias pessoas.
- C4 Muitas pessoas assassinadas.

## Frequência e Tempo de Exposição

- F1 Exposição de rara a freqüente na região de perigo.
- F2 Exposição de freqüente a permanente na região de perigo.

## Possibilidade de evitar o evento

- P1 Possível sob certas condições.
- P2 Quase impossível.

## Probabilidade de ocorrências indesejáveis

- W1 Probabilidade muito pequena de que as ocorrências aconteçam.
- W2 Probabilidade pequena de que as ocorrências aconteçam.
- W3 Probabilidade relativamente alta de que as ocorrências aconteçam.

```
void IniciaBAC (void)
```

```
{
 QFTxBAC = 0;
 PFTxBAC = 0;
 AFTxBAC = 0;
 QFTxBAC = 0;
 UFRxBAC = 0;
 PFRxBAC = 0;
 QFRxBAC = 0;
 FlagTxBAC = 0;
 FlagEstadoBAC = 0;
 FlagCpcBAC = 0;
 FalhasBAC = 0;
 SemaforoTxBAC = 0x0;
 TimerTimeoutBAC = TimerRelogio3;
 ProgramaRS232C();
 if ((PFRxBAC >= MAXFRxBAC) ||
 (UFRxBAC >= MAXURxBAC) ||
 (QFRxBAC >= MAXQRxBAC))
 {
 disable();
 if ((PFRxBAC >= MAXFRxBAC) ||
 (UFRxBAC >= MAXURxBAC) ||
 (QFRxBAC >= MAXQRxBAC))
 {
 return;
 }
 }
 /* Copia a mensagem para a fila. */
 i = 0;
 while (i < n)
 {
 BYTE i;
 /* [01] */
 if ((SemaforoTxBAC == 0) && (QFTxBAC < MAXFTXBAC) &&
 (n < MAXBFTXBAC))
 {
 /* Copia a mensagem para a fila. */
 i = 0;
 while (i < n)
 {

```

```
FrxBAC[UFrxBAC] = FrxBAC[i];
FrxBAC[UFrxBAC] = FrxBAC[i];
FrxBAC[UFrxBAC] = FrxBAC[i];
/* Atualiza índices da fila de transmissão. */
if (UFrxBAC == (MAXFTXBAC-1))
 UFrxBAC = 0;
else
 UFrxBAC++;
FrxBAC[UFrxBAC] = FrxBAC[i];
FrxBAC[UFrxBAC] = FrxBAC[i];
FrxBAC[UFrxBAC] = FrxBAC[i];
/* Existe mensagens na Fila de Recepcao. */
/* Sim. Copia a mensagem para o ponteiro. */
i = 0;
while (i < FrxBAC[PFRxBAC].m[0])
{
 msg[i] = FrxBAC[PFRxBAC].m[i+1];
 i++;
}
/* Atualiza índices da fila de recepcao. */
if (PFRxBAC == (MAXPFRxBAC-1))
 PFRxBAC = 0;
else
 PFRxBAC++;
FrxBAC[PFRxBAC] = FrxBAC[i];
FrxBAC[PFRxBAC] = FrxBAC[i];
FrxBAC[PFRxBAC] = FrxBAC[i];
void EnviaAckBAC (void)
{
 TxMsgAckBAC();
 FlagTxBAC = 2;
 TimerTimeoutBAC = TimerRelogio3;
 return;
}
void EnviaNackBAC (void)
```

# IEC 61508 - Determinar Redução de Risco e SIL

|    |    |    | W3 | W2 | W1 |
|----|----|----|----|----|----|
| C2 | C1 |    | A  | -  | -  |
|    | F1 | P1 | B  | A  | -  |
|    |    | P2 | C  | B  | A  |
|    | F2 | P1 | D  | C  | B  |
| C3 |    | P2 | E  | D  | C  |
|    | F1 |    | F  | E  | D  |
|    | F2 |    | G  | F  | E  |
|    | C4 |    | H  | G  | F  |

SIL Falhas inseguras / hora

|   |                                 |
|---|---------------------------------|
| 4 | $10^{-9} < \lambda_u < 10^{-8}$ |
| 3 | $10^{-8} < \lambda_u < 10^{-7}$ |
| 2 | $10^{-7} < \lambda_u < 10^{-6}$ |
| 1 | $10^{-6} < \lambda_u < 10^{-5}$ |

Integridade de Segurança

Requisitos de Segurança

Requisitos Especiais de Segurança

SIL 1

SIL 2

SIL 3

SIL 4

Sistemas de Proteção não Suficientes

# IEC 61508 - Determinar Redução de Risco e SIL

|    |                  | W3 | W2 | W1 |     |                                 |
|----|------------------|----|----|----|-----|---------------------------------|
| C1 | FlagTxBAC = 0;   | A  | -  | -  | SIL | Falhas inseguras / hora         |
| F1 | FlagCpCBAC = 0;  | B  | A  | -  | 4   | $10^{-9} < \lambda_u < 10^{-8}$ |
| C2 | NrtxBAC = 0;     | C  | B  | A  | 3   | $10^{-8} < \lambda_u < 10^{-7}$ |
| F2 | SemaforCBAC = 0; | D  | C  | B  |     |                                 |

## Redução de Risco      Nível de Integridade de Segurança

|      |                                       |
|------|---------------------------------------|
| -    | Sem Requisitos de Segurança           |
| A    | Sem Requisitos Especiais de Segurança |
| B, C | SIL 1                                 |
| D    | SIL 2                                 |
| E, F | SIL 3                                 |
| G    | SIL 4                                 |
| H    | Sistemas de Proteção não Suficientes  |

# IEC 61508 - Determinar Redução de Risco e SIL

|    | W3 | W2 | W1 |
|----|----|----|----|
| C1 | A  | -  | -  |
| F1 | B  | A  | -  |
| F2 | C  | B  | A  |
|    | D  | C  | B  |
|    | E  | D  | C  |
| F1 | F  | E  | D  |
| F2 | G  | F  | E  |
| C4 | H  | G  | F  |

| <b>SIL</b> | <b>Falhas inseguras / hora</b>  |
|------------|---------------------------------|
| <b>4</b>   | $10^{-9} < \lambda_u < 10^{-8}$ |
| <b>3</b>   | $10^{-8} < \lambda_u < 10^{-7}$ |
| <b>2</b>   | $10^{-7} < \lambda_u < 10^{-6}$ |
| <b>1</b>   | $10^{-6} < \lambda_u < 10^{-5}$ |

## Redução de Risco

## Nível de Integridade de Segurança

# Sem Requisitos de Segurança

# Sem Requisitos Especiais de Segurança

# SIL 1

## SIL 2

### SIL 3

# SIL 4

## Sistemas de Proteção não Suficientes

# Software & Segurança: EN50128 SWSIL

5.2.2 Os requisitos de nível de integridade de segurança de software (SWSIL) deverão ser decididos com base no nível de risco associado com a **utilização do software no sistema** e no nível de integridade de segurança de software.

[CENELEC 50128]

Tendo caracterizado o Nível de Integridade de Segurança (SIL) para o Sistema Metro-ferroviário, o Software na área de Sinalização para tal Sistema deverá possuir o mesmo valor.

```
void IniciaBAC (void)
```

```
{
 UFLxBAC = 0;
 PFLxBAC = 0;
 AFLxBAC = 0;
 QFLxBAC = 0;
 UFRxBAC = 0;
```

```
/* [01] */
```

```
if ((SemaforoTxBAC == 0) && (QFLxBAC < MAXFTXBAC) &&
 (n < MAXBFTXBAC))
```

```
{
 /* Copia a mensagem para a fila. */
 i = 0;
 while (i < n)
 {
```

```
FLxBAC[UFLxBAC].m[i+1] = 0;
 FLxBAC[UFLxBAC].m[i+2] = 0;
 FLxBAC[UFLxBAC].m[i+3] = 0;
```

```
FLxBAC[UFLxBAC].m[i+4] = 0;
```

```
/* Adiciona mensagens da fila de transmissao. */
```

```
if (UFLxBAC < MAXFTXBAC-1)
```

```
{
 UFLxBAC = 0;
```

```
 UFLxBAC++;
 UFLxBAC++;
 UFLxBAC++;
 UFLxBAC++;
 return(TRUE);
}
```

```
return(FALSE);
}
```

```
BYTE ObtemMsgBAC (BYTE *msg)
```

```
{
 BYTE i;
```

```
/* Existe mensagens na Fila de Recepcao. */
```

```
if (QFLxBAC > 0)
```

```
{
 /* Sim. Copia a mensagem para o ponteiro. */
```

```
i = 0;
```

```
while (i < n)
```

```
{
 msg[i] = FLxBAC[PFLxBAC].m[i+1];
 i++;
 PFLxBAC++;
 PFLxBAC++;
 PFLxBAC++;
 PFLxBAC++;
 return(msg);
}
```

```
return(0);
```

```
void EnviaAckBAC (void)
```

```
{
 TxMsgAckBAC();
 FlagTxBAC = 2;
 TimerTimeoutBAC = TimerRelogio3;
 return;
```

```
}
```

```
void EnviaNackBAC (void)
```

```
{
 TxMsgNackBAC();
 FlagTxBAC = 1;
 TimerTimeoutBAC = TimerRelogio3;
 return;
```

```
}
```

```
void IniciaBAC (void)
```

```
{
 UFLxBAC = 0;
 PFLxBAC = 0;
 AFLxBAC = 0;
 QFLxBAC = 0;
 UFRxBAC = 0;
```

```
/* [01] */
```

```
if ((SemaforoTxBAC == 0) && (QFLxBAC < MAXFTXBAC) &&
 (n < MAXBFTXBAC))
```

```
{
 /* Copia a mensagem para a fila. */
```

```
i = 0;
```

```
while (i < n)
```

```
{
 /* Adiciona mensagens da fila de recepcao. */
```

```
if (UFLxBAC < MAXFTXBAC-1)
```

```
{
 UFLxBAC = 0;
```

```
 UFLxBAC++;
 UFLxBAC++;
 UFLxBAC++;
 UFLxBAC++;
 return(TRUE);
}
```

```
return(FALSE);
}
```

```
BYTE ObtemMsgBAC (BYTE *msg)
```

```
{
 BYTE i;
```

```
/* Existe mensagens na Fila de Recepcao. */
```

```
if (QFLxBAC > 0)
```

```
{
 /* Sim. Copia a mensagem para o ponteiro. */
```

```
i = 0;
```

```
while (i < n)
```

```
{
 msg[i] = FLxBAC[PFLxBAC].m[i+1];
 i++;
 PFLxBAC++;
 PFLxBAC++;
 PFLxBAC++;
 PFLxBAC++;
 return(msg);
}
```

```
return(0);
```

```
void EnviaAckBAC (void)
```

```
{
 TxMsgAckBAC();
 FlagTxBAC = 2;
 TimerTimeoutBAC = TimerRelogio3;
 return;
```

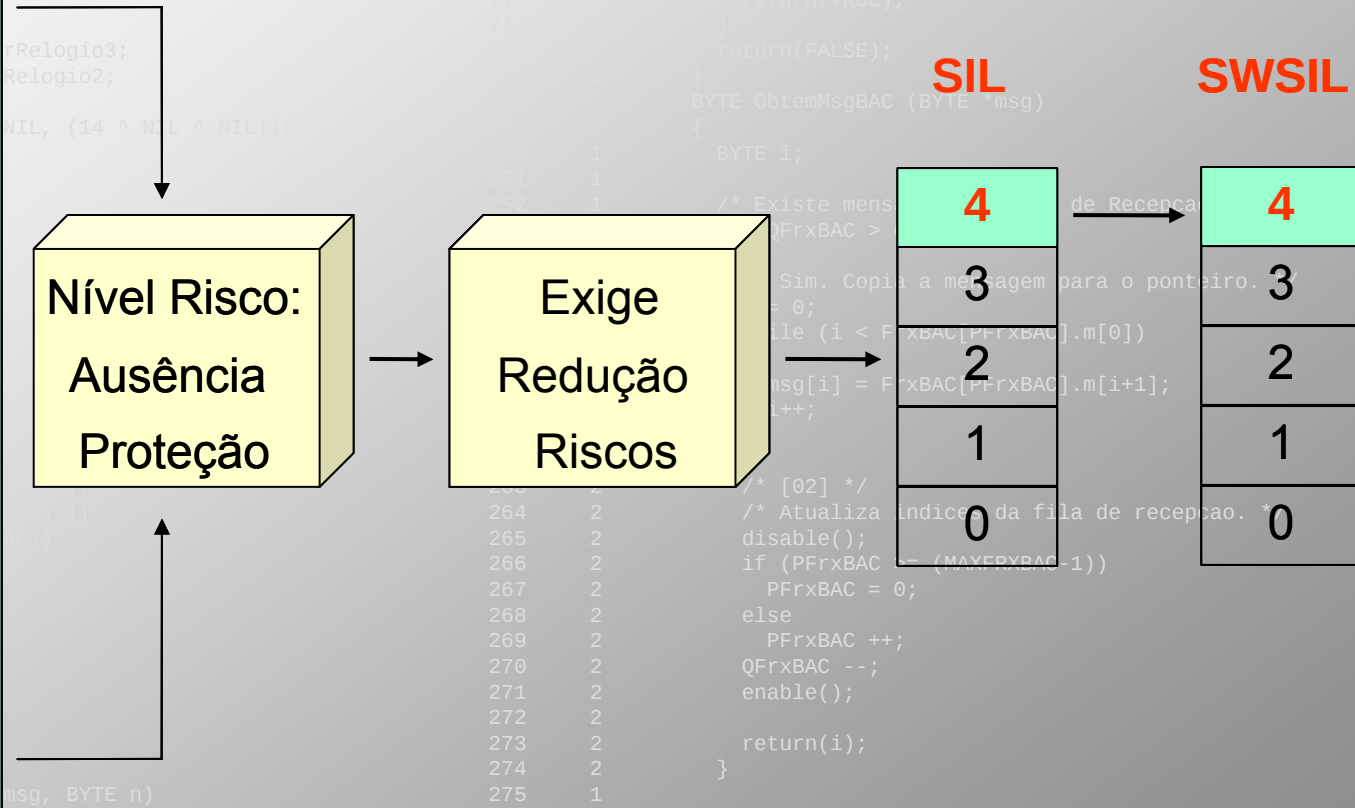
```
}
```

```
void EnviaNackBAC (void)
```

```
{
 TxMsgNackBAC();
 FlagTxBAC = 1;
 TimerTimeoutBAC = TimerRelogio3;
 return;
```

```
}
```

Conseqüência  
Ameaça



```

 FtxBAC[FTxBAC].m[i+1] = 0;
}

FTxBAC[UFtxBAC] = 0;

/* Atualiza o ponteiro da fila de transmissao. */
if (FTxBAC == (MAXFTxBAC-1))
 FTxBAC = 0;
else
 FTxBAC++;

/* Atualiza o ponteiro da fila de recepcao. */
if (PFrxBAC == (MAXPFrxBAC-1))
 PFrxBAC = 0;
else
 PFrxBAC++;

return(TRUE);
}

return(FALSE);
}

BYTE ObtemMsgBAC (BYTE *msg)
{
 BYTE i;

 /* Existe mensagem na fila de recepcao? */
 if (PFrxBAC > QFrxBAC)
 return(0);

 /* Sim. Copia a mensagem para o ponteiro. */
 *msg = 0;
 while (i < PFrxBAC[PFrxBAC].m[0])
 msg[i] = FtxBAC[PFrxBAC].m[i+1];
 i++;

 /* [02] */
 /* Atualiza o indice da fila de recepcao. */
 disable();
 if (PFrxBAC == (MAXPFrxBAC-1))
 PFrxBAC = 0;
 else
 PFrxBAC++;
 QFrxBAC--;
 enable();

 return(i);
}

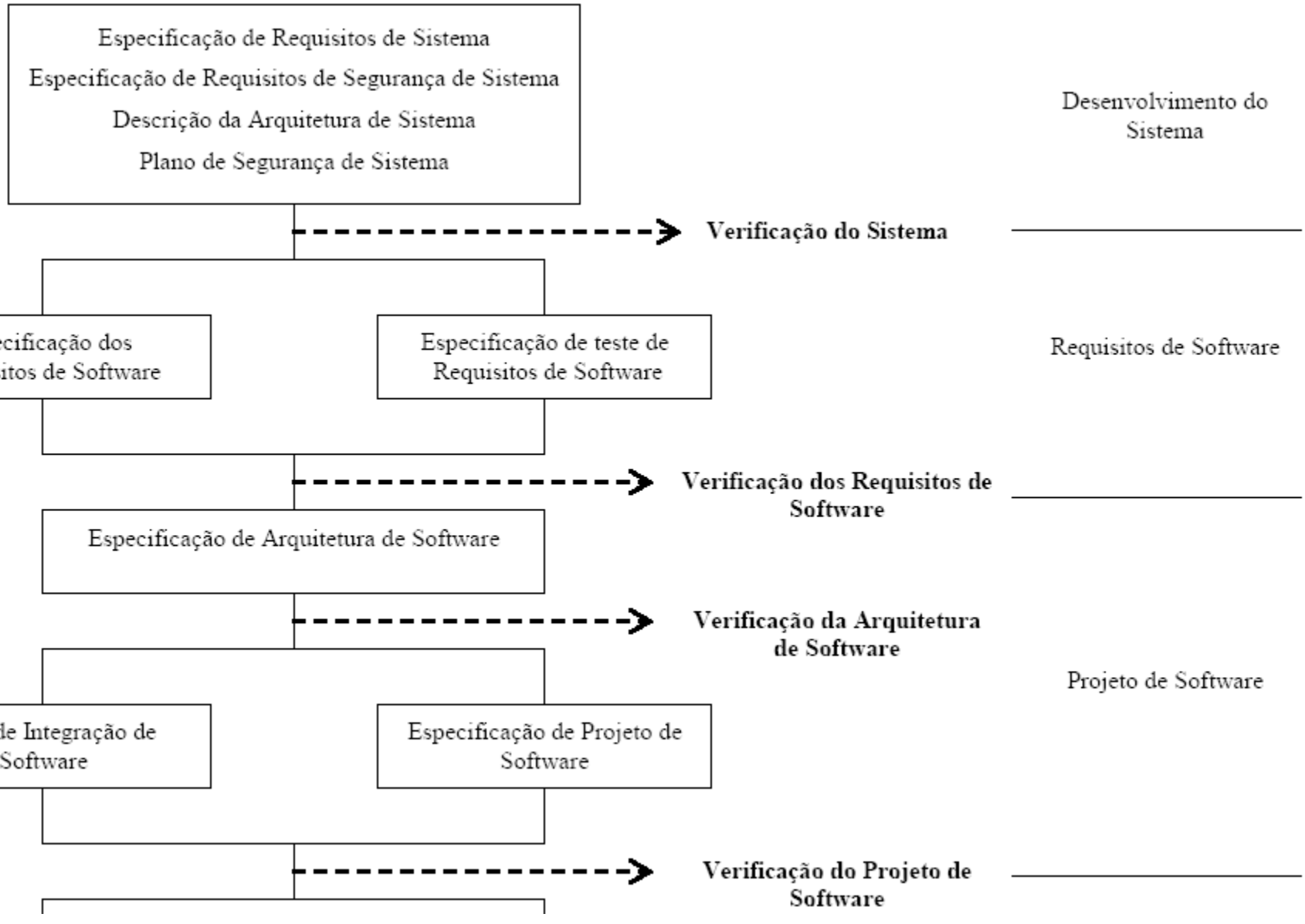
return(0);
}

void EnviaAckBAC (void)
{
 TxMsgAckBAC();
 FlagTxBAC = 2;
 TimerTimeoutBAC = TimerRelogio3;
 return;
}

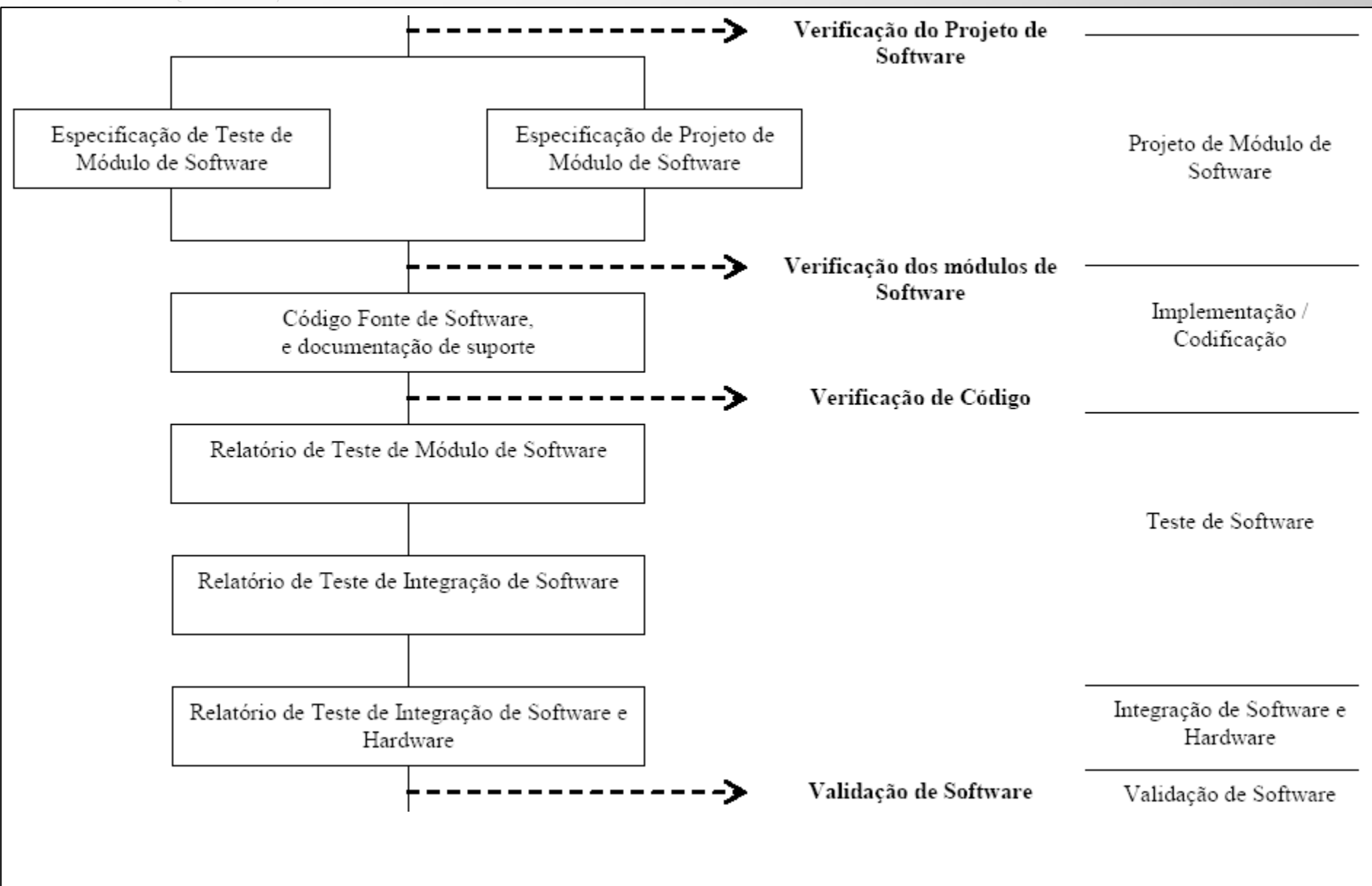
void EnviaNackBAC (void)

```

# CENELEC 50128 - Ciclo de Vida I



# CENELEC 50128 - Ciclo de Vida II



# CENEL



## Requisito Técnico de Software - **Segurança**

Os conceitos de “**Fail Safe**” devem ser utilizados no Software relacionado às funções de Segurança Crítica ou Vitais.

Deve atender ao “**Software Safety Integrity Level**” (SIL) **quatro (4)**, conforme a norma IEC 61508 “Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems”.

Devem ser atendidas as recomendações da norma **CENELEC 50128** “Railway applications – Communications, Signaling and Processing Systems – Software for Railway Control and Protection Systems”.

[EPS 2004]

```
void IniciaBAC (void)
{
 UFTxBAC = 0;
 PFTxBAC = 0;
 AFTxBAC = 0;
 QFTxBAC = 0;
 UFRxBAC = 0;
 PFRxBAC = 0;
 QFRxBAC = 0;
 FlagTxBAC = 0;
 FlagCpcBAC = 0;
 SemaforoTxBAC = 0x0F;
 TimerTimeoutBAC = TimerRelogio3;
 TimerJanelaBAC = TimerRelogio3;
 InserirFila(14, NIL, NIL, (14 * NIL * NIL));
 return;
}

BYTE EnviaMsgBAC (BYTE *msg, BYTE n)
{
 BYTE i;

 /* [01] */
 if ((SemaforoTxBAC == 0) && (QFTxBAC < MAXFTXBAC) &&
 (n < MAXBFTXBAC))
 {
 /* Copia a mensagem para a fila. */
 i = 0;
 while (i < n)
 {
```

```
 FrxBAC[UFTxBAC].m[i] = msg[i];
 UFTxBAC++;
 if (UFTxBAC == MAXFTXBAC)
 {
 /* Atualiza índices da fila de transmissão. */
 UFTxBAC = 0;
 PFTxBAC = 0;
 AFTxBAC = 0;
 QFTxBAC = 0;
 UFRxBAC = 0;
 PFRxBAC = 0;
 QFRxBAC = 0;
 FlagTxBAC = 0;
 FlagCpcBAC = 0;
 SemaforoTxBAC = 0x0F;
 TimerTimeoutBAC = TimerRelogio3;
 TimerJanelaBAC = TimerRelogio3;
 InserirFila(14, NIL, NIL, (14 * NIL * NIL));
 return;
 }
 /* Copia a mensagem para o ponteiro. */
 i = 0;
 while (i < FrxBAC[PFRxBAC].m[0])
 {
 msg[i] = FrxBAC[PFRxBAC].m[i+1];
 i++;
 }
 /* [02] */
 /* Atualiza índices da fila de recepcao. */
 if (PFRxBAC >= (MAXFRXBAC-1))
 {
 PFRxBAC = 0;
 PFRxBAC++;
 QFRxBAC--;
 enable();
 return(i);
 }
 return(0);
 }
 }

void EnviaAckBAC (void)
{
 TxMsgAckBAC();
 FlagTxBAC = 2;
 TimerTimeoutBAC = TimerRelogio3;
 return;
}

void EnviaNackBAC (void)
```

# Emprego de Software em Sinalização - Horizonte

## Considerações:

- Uso Intensivo de SW em Tempo Real e Segurança Crítica;
- Fatores de aumento natural do risco;
- Transmissão por meio aberto de Sinais Vitais;
- Determinação de distância por múltiplos sensores;
- Emprego limitado de Hardware fail-safe;

Os **conceitos consagrados** de Segurança de um Sistema de Sinalização e Controle devem ser implementados de forma **completa e claramente verificáveis**. [IEEE 1474-1]

```

89 void IniciaBAC (void)
90 {
91 UFTxBAC = 0;
92 PFTxBAC = 0;
93 AFTxBAC = 0;
94 QFTxBAC = 0;
95 UFRxBAC = 0;
96 PFRxBAC = 0;
97 QFRxBAC = 0;
98 FlagEstadoBAC = 0;
99 FlagCpcBAC = 0;
100 FalhasBAC = 0;
101 NrtxBAC = MAXRETXBAC;
102 TimerTimeoutBAC = TimerRelogio3;
103 TimerJanelaBAC = TimerRelogio2;
104 ProgramaRS232C();
105 if (MESTREBAC == 1)
106 {
107 return;
108 }
109 Diagnostico[0x44]++;
110 if ((PFRxBAC >= MAXFRXBAC) || (UFRxBAC >= MAXFRXBAC) || (QFRxBAC >= MAXFRXBAC))
111 {
112 /* Atualiza indices da fila de recepcao. */
113 disable();
114 if (PFRxBAC >= (MAXFRXBAC-1))
115 PFRxBAC = 0;
116 else
117 PFRxBAC ++;
118 QFRxBAC --;
119 enable();
120 return(1);
121 }
122 return(0);
123 }
124 void EnviaAckBAC (void)
125 {
126 TxMsgAckBAC();
127 FlagTxBAC = 2;
128 TimerTimeoutBAC = TimerRelogio3;
129 return;
130 }
131 void EnviaNackBAC (void)
132 {
133 TxMsgAckBAC();
134 FlagTxBAC = 2;
135 TimerTimeoutBAC = TimerRelogio3;
136 return;
137 }

```

```

202 3 FtxBAC[UFTxBAC] = 0;
203 3
204 2 FtxBAC[PFTxBAC] = 0;
205 2
206 2 FtxBAC[AFTxBAC] = 0;
207 2
208 2 FtxBAC[QFTxBAC] = 0;
209 2
210 2 /* Atualiza indices da fila de transmissao. */
211 2 if (UFTxBAC >= (MAXFTXBAC-1))
212 2 UFTxBAC = 0;
213 2 else
214 2 UFTxBAC ++;
215 2
216 2 PFTxBAC --;
217 2
218 2 AFTxBAC ++;
219 2
220 2 QFTxBAC ++;
221 2
222 2
223 2
224 2
225 2
226 2
227 2
228 2
229 2
230 2
231 2
232 2
233 2
234 2
235 2
236 2
237 2
238 2
239 2
240 2
241 2
242 2
243 2
244 2
245 2
246 2
247 2
248 2
249 2
250 2
251 2
252 2
253 2
254 2
255 2
256 2
257 2
258 2
259 2
260 3
261 3
262 2
263 2
264 2
265 2
266 2
267 2
268 2
269 2
270 2
271 2
272 2
273 2
274 2
275 2
276 2
277 1
278 1
279 1
280 1
281 1
282 1
283 1
284 1
285 1
286 1
287 1
288 1
289 1
290 1
291 1
292 1
293 1
294 1
295 1
296 1
297 1
298 1
299 1
300 1
301 1
302 1
303 1
304 1
305 1
306 1
307 1
308 1
309 1
310 1
311 1
312 1
313 1
314 1
315 1
316 1
317 1
318 1
319 1
320 1
321 1
322 1
323 1
324 1
325 1
326 1
327 1
328 1
329 1
330 1
331 1
332 1
333 1

```

## O Software com Qualidade:

- Maior Garantia do Nível de Segurança
- Redução do Retrabalho
- Otimização de Recursos
- Controle sobre o Processo
- Controle sobre Tecnologias em Implantação
- Integra Experiência e Capacidade de Inovação
- Incorporação de Iniciativas dos maiores Consumidores de Software mundiais e da Produção Científico-acadêmica na área.

```
void IniciaBAC (void)
```

```
{
 Diagnostico[0] = 0;
 UFTxBAC = 0;
 PFTxBAC = 0;
 AFTxBAC = 0;
 QFTxBAC = 0;
 UFRxBAC = 0;
 PFRxBAC = 0;
 QFRxBAC = 0;
 NrtxBAC = 0;
 SemaforoTxBAC = 0;
 FlagCpcBAC = 0;
 FalhasBAC = 0;
 NrtxBAC = MAXRETXBAC;
 SemaforoTxBAC = 0;
 TimerRelogioBAC = TimerRelogio3;
 ProgramaRS232C();
 InsereTarefa(14, NIL, NIL, (14 ^ NIL ^ NIL));
 return;
}

void TestaFilaBAC (void)
{
 disable();
 if ((PFRxBAC >= MAXFRXBAC) ||
 (QFRxBAC >= MAXQFRXBAC))
 {
 enable();
 return;
 }
 /* Copia a mensagem para a fila. */
 i = 0;
 while (i < n)
 {
 msg[i] = FrxBAC[PFRxBAC].m[i+1];
 i++;
 }
 /* Atualiza indices da fila de recepcao. */
 disable();
 if (PFRxBAC >= (MAXFRXBAC-1))
 {
 PFRxBAC = 0;
 }
 if (QFRxBAC >= (MAXQFRXBAC-1))
 {
 QFRxBAC = 0;
 }
 enable();
 return;
}

void EnviaAckBAC (void)
{
 /* Copia a mensagem para a fila. */
 i = 0;
 while (i < n)
 {
 msg[i] = FrxBAC[PFRxBAC].m[i+1];
 i++;
 }
 /* Atualiza indices da fila de recepcao. */
 disable();
 if (PFRxBAC >= (MAXFRXBAC-1))
 {
 PFRxBAC = 0;
 }
 if (QFRxBAC >= (MAXQFRXBAC-1))
 {
 QFRxBAC = 0;
 }
 enable();
 return;
}

void EnviaNackBAC (void)
```

```

89 void IniciaBAC (void)
90 {
91 1 Diagnostico[0x07] ++;
92 1 UFTxBAC = 0;
93 1 PFTxBAC = 0;
94 1 AFTxBAC = 0;
95 1 QFTxBAC = 0;
96 1 UFRxBAC = 0;
97 1 PFRxBAC = 0;
98 1 QFRxBAC = 0;
99 1 FlagTxBAC = 0;
100 1 FlagEstadoBAC = 0;
101 1 FlagCpcBAC = 0;
102 1 FalhasBAC = 0;
103 1 NrtxBAC = MAXRETXBAC;
104 1 SemaforoTxBAC[0x0F] = 0;
105 1 disable();
106 1 enable();
107 1 ProgramaRS232C();
108 1 InsereTarefa(14, NIL, NIL, (14 ^ NIL ^ NIL));
109 1 return;
110 }

113 void TestaFilaBAC (void)
114 {
115 1 Diagnostico[0x44]++;
116 1 disable();
117 1 if ((PFRxBAC >= MAXFRXBAC) ||
118 1 (UFRxBAC >= MAXFRXBAC) ||
119 1 (QFRxBAC >= MAXFRXBAC) ||
120 1 (PFTxBAC >= MAXFTXBAC) ||
121 1 (UFTxBAC >= MAXFTXBAC) ||
122 1 (QFTxBAC >= MAXFTXBAC))
123 2 {
124 2 /* Preenche a fila de recepcao. */
125 2 while (i < 0x09;
126 2 i++)
127 2 {
128 2 BYTE i;
129 2 /* [01] */
130 2 if ((SemaforoTxBAC == 0) && (QFTxBAC < MAXFTXBAC) &&
131 2 (n < MAXBFTXBAC))
132 2 {
133 2 /* Copia a mensagem para a fila. */
134 2 i = 0;
135 2 while (i < n)
136 2 {
137 2 BYTE i;
138 2 /* [02] */
139 2 /* Atualiza indices da fila de recepcao. */
140 2 disable();
141 2 if (PFRxBAC >= (MAXFRXBAC-1))
142 2 {
143 2 PFRxBAC = 0;
144 2 }
145 2 else
146 2 {
147 2 PFRxBAC ++;
148 2 }
149 2 QFRxBAC --;
150 2 enable();
151 2 return(i);
152 2 }
153 2 return(0);
154 2 }
155 2 void EnviaAckBAC (void)
156 2 {
157 2 TxMsgAckBAC();
158 2 FlagTxBAC = 2;
159 2 TimerTimeoutBAC = TimerRelogio3;
160 2 return;
161 2 }
162 2 void EnviaNackBAC (void)
163 2 {
164 2 TxMsgAckBAC();
165 2 FlagTxBAC = 2;
166 2 TimerTimeoutBAC = TimerRelogio3;
167 2 return;
168 2 }
169 2 }
170 2 }
171 2 }
172 2 }
173 2 }
174 2 }
175 2 }
176 2 }
177 2 }
178 2 }
179 2 }
180 2 }
181 2 }
182 2 }
183 2 }
184 2 }
185 2 }
186 2 }
187 2 }
188 2 }
189 2 }
190 2 }
191 2 }
192 2 }
193 2 }
194 2 }
195 2 }
196 2 }
197 2 }
198 2 }
199 2 }
200 2 }
201 2 }
202 2 }
203 2 }
204 2 }
205 2 }
206 2 }
207 2 }
208 2 }
209 2 }
210 2 }
211 2 }
212 2 }
213 2 }
214 2 }
215 2 }
216 2 }
217 2 }
218 2 }
219 2 }
220 2 }
221 2 }
222 2 }
223 2 }
224 2 }
225 2 }
226 2 }
227 2 }
228 2 }
229 2 }
230 2 }
231 2 }
232 2 }
233 2 }
234 2 }
235 2 }
236 2 }
237 2 }
238 2 }
239 2 }
240 2 }
241 2 }
242 2 }
243 2 }
244 2 }
245 2 }
246 2 }
247 2 }
248 2 }
249 2 }
250 2 }
251 2 }
252 2 }
253 2 }
254 2 }
255 2 }
256 2 }
257 2 }
258 2 }
259 2 }
260 2 }
261 2 }
262 2 }
263 2 }
264 2 }
265 2 }
266 2 }
267 2 }
268 2 }
269 2 }
270 2 }
271 2 }
272 2 }
273 2 }
274 2 }
275 2 }
276 2 }
277 2 }
278 2 }
279 2 }
280 2 }
281 2 }
282 2 }
283 2 }
284 2 }
285 2 }
286 2 }
287 2 }
288 2 }
289 2 }
290 2 }
291 2 }
292 2 }
293 2 }
294 2 }
295 2 }
296 2 }
297 2 }
298 2 }
299 2 }
300 2 }
301 2 }
302 2 }
303 2 }
304 2 }
305 2 }
306 2 }
307 2 }
308 2 }
309 2 }
310 2 }
311 2 }
312 2 }
313 2 }
314 2 }
315 2 }
316 2 }
317 2 }
318 2 }
319 2 }
320 2 }
321 2 }
322 2 }
323 2 }
324 2 }
325 2 }
326 2 }
327 2 }
328 2 }
329 2 }
330 2 }
331 2 }
332 2 }
333 2 }
334 2 }
335 2 }
336 2 }
337 2 }
338 2 }
339 2 }
340 2 }
341 2 }
342 2 }
343 2 }
344 2 }
345 2 }
346 2 }
347 2 }
348 2 }
349 2 }
350 2 }
351 2 }
352 2 }
353 2 }
354 2 }
355 2 }
356 2 }
357 2 }
358 2 }
359 2 }
360 2 }
361 2 }
362 2 }
363 2 }
364 2 }
365 2 }
366 2 }
367 2 }
368 2 }
369 2 }
370 2 }
371 2 }
372 2 }
373 2 }
374 2 }
375 2 }
376 2 }
377 2 }
378 2 }
379 2 }
380 2 }
381 2 }
382 2 }
383 2 }
384 2 }
385 2 }
386 2 }
387 2 }
388 2 }
389 2 }
390 2 }
391 2 }
392 2 }
393 2 }
394 2 }
395 2 }
396 2 }
397 2 }
398 2 }
399 2 }
400 2 }
401 2 }
402 2 }
403 2 }
404 2 }
405 2 }
406 2 }
407 2 }
408 2 }
409 2 }
410 2 }
411 2 }
412 2 }
413 2 }
414 2 }
415 2 }
416 2 }
417 2 }
418 2 }
419 2 }
420 2 }
421 2 }
422 2 }
423 2 }
424 2 }
425 2 }
426 2 }
427 2 }
428 2 }
429 2 }
430 2 }
431 2 }
432 2 }
433 2 }
434 2 }
435 2 }
436 2 }
437 2 }
438 2 }
439 2 }
440 2 }
441 2 }
442 2 }
443 2 }
444 2 }
445 2 }
446 2 }
447 2 }
448 2 }
449 2 }
450 2 }
451 2 }
452 2 }
453 2 }
454 2 }
455 2 }
456 2 }
457 2 }
458 2 }
459 2 }
460 2 }
461 2 }
462 2 }
463 2 }
464 2 }
465 2 }
466 2 }
467 2 }
468 2 }
469 2 }
470 2 }
471 2 }
472 2 }
473 2 }
474 2 }
475 2 }
476 2 }
477 2 }
478 2 }
479 2 }
480 2 }
481 2 }
482 2 }
483 2 }
484 2 }
485 2 }
486 2 }
487 2 }
488 2 }
489 2 }
490 2 }
491 2 }
492 2 }
493 2 }
494 2 }
495 2 }
496 2 }
497 2 }
498 2 }
499 2 }
500 2 }
501 2 }
502 2 }
503 2 }
504 2 }
505 2 }
506 2 }
507 2 }
508 2 }
509 2 }
510 2 }
511 2 }
512 2 }
513 2 }
514 2 }
515 2 }
516 2 }
517 2 }
518 2 }
519 2 }
520 2 }
521 2 }
522 2 }
523 2 }
524 2 }
525 2 }
526 2 }
527 2 }
528 2 }
529 2 }
530 2 }
531 2 }
532 2 }
533 2 }
534 2 }
535 2 }
536 2 }
537 2 }
538 2 }
539 2 }
540 2 }
541 2 }
542 2 }
543 2 }
544 2 }
545 2 }
546 2 }
547 2 }
548 2 }
549 2 }
550 2 }
551 2 }
552 2 }
553 2 }
554 2 }
555 2 }
556 2 }
557 2 }
558 2 }
559 2 }
560 2 }
561 2 }
562 2 }
563 2 }
564 2 }
565 2 }
566 2 }
567 2 }
568 2 }
569 2 }
570 2 }
571 2 }
572 2 }
573 2 }
574 2 }
575 2 }
576 2 }
577 2 }
578 2 }
579 2 }
580 2 }
581 2 }
582 2 }
583 2 }
584 2 }
585 2 }
586 2 }
587 2 }
588 2 }
589 2 }
590 2 }
591 2 }
592 2 }
593 2 }
594 2 }
595 2 }
596 2 }
597 2 }
598 2 }
599 2 }
600 2 }
601 2 }
602 2 }
603 2 }
604 2 }
605 2 }
606 2 }
607 2 }
608 2 }
609 2 }
610 2 }
611 2 }
612 2 }
613 2 }
614 2 }
615 2 }
616 2 }
617 2 }
618 2 }
619 2 }
620 2 }
621 2 }
622 2 }
623 2 }
624 2 }
625 2 }
626 2 }
627 2 }
628 2 }
629 2 }
630 2 }
631 2 }
632 2 }
633 2 }
634 2 }
635 2 }
636 2 }
637 2 }
638 2 }
639 2 }
640 2 }
641 2 }
642 2 }
643 2 }
644 2 }
645 2 }
646 2 }
647 2 }
648 2 }
649 2 }
650 2 }
651 2 }
652 2 }
653 2 }
654 2 }
655 2 }
656 2 }
657 2 }
658 2 }
659 2 }
660 2 }
661 2 }
662 2 }
663 2 }
664 2 }
665 2 }
666 2 }
667 2 }
668 2 }
669 2 }
670 2 }
671 2 }
672 2 }
673 2 }
674 2 }
675 2 }
676 2 }
677 2 }
678 2 }
679 2 }
680 2 }
681 2 }
682 2 }
683 2 }
684 2 }
685 2 }
686 2 }
687 2 }
688 2 }
689 2 }
690 2 }
691 2 }
692 2 }
693 2 }
694 2 }
695 2 }
696 2 }
697 2 }
698 2 }
699 2 }
700 2 }
701 2 }
702 2 }
703 2 }
704 2 }
705 2 }
706 2 }
707 2 }
708 2 }
709 2 }
710 2 }
711 2 }
712 2 }
713 2 }
714 2 }
715 2 }
716 2 }
717 2 }
718 2 }
719 2 }
720 2 }
721 2 }
722 2 }
723 2 }
724 2 }
725 2 }
726 2 }
727 2 }
728 2 }
729 2 }
730 2 }
731 2 }
732 2 }
733 2 }
734 2 }
735 2 }
736 2 }
737 2 }
738 2 }
739 2 }
740 2 }
741 2 }
742 2 }
743 2 }
744 2 }
745 2 }
746 2 }
747 2 }
748 2 }
749 2 }
750 2 }
751 2 }
752 2 }
753 2 }
754 2 }
755 2 }
756 2 }
757 2 }
758 2 }
759 2 }
760 2 }
761 2 }
762 2 }
763 2 }
764 2 }
765 2 }
766 2 }
767 2 }
768 2 }
769 2 }
770 2 }
771 2 }
772 2 }
773 2 }
774 2 }
775 2 }
776 2 }
777 2 }
778 2 }
779 2 }
780 2 }
781 2 }
782 2 }
783 2 }
784 2 }
785 2 }
786 2 }
787 2 }
788 2 }
789 2 }
790 2 }
791 2 }
792 2 }
793 2 }
794 2 }
795 2 }
796 2 }
797 2 }
798 2 }
799 2 }
800 2 }
801 2 }
802 2 }
803 2 }
804 2 }
805 2 }
806 2 }
807 2 }
808 2 }
809 2 }
810 2 }
811 2 }
812 2 }
813 2 }
814 2 }
815 2 }
816 2 }
817 2 }
818 2 }
819 2 }
820 2 }
821 2 }
822 2 }
823 2 }
824 2 }
825 2 }
826 2 }
827 2 }
828 2 }
829 2 }
830 2 }
831 2 }
832 2 }
833 2 }
834 2 }
835 2 }
836 2 }
837 2 }
838 2 }
839 2 }
840 2 }
841 2 }
842 2 }
843 2 }
844 2 }
845 2 }
846 2 }
847 2 }
848 2 }
849 2 }
850 2 }
851 2 }
852 2 }
853 2 }
854 2 }
855 2 }
856 2 }
857 2 }
858 2 }
859 2 }
860 2 }
861 2 }
862 2 }
863 2 }
864 2 }
865 2 }
866 2 }
867 2 }
868 2 }
869 2 }
870 2 }
871 2 }
872 2 }
873 2 }
874 2 }
875 2 }
876 2 }
877 2 }
878 2 }
879 2 }
880 2 }
881 2 }
882 2 }
883 2 }
884 2 }
885 2 }
886 2 }
887 2 }
888 2 }
889 2 }
890 2 }
891 2 }
892 2 }
893 2 }
894 2 }
895 2 }
896 2 }
897 2 }
898 2 }
899 2 }
900 2 }
901 2 }
902 2 }
903 2 }
904 2 }
905 2 }
906 2 }
907 2 }
908 2 }
909 2 }
910 2 }
911 2 }
912 2 }
913 2 }
914 2 }
915 2 }
916 2 }
917 2 }
918 2 }
919 2 }
920 2 }
921 2 }
922 2 }
923 2 }
924 2 }
925 2 }
926 2 }
927 2 }
928 2 }
929 2 }
930 2 }
931 2 }
932 2 }
933 2 }
934 2 }
935 2 }
936 2 }
937 2 }
938 2 }
939 2 }
940 2 }
941 2 }
942 2 }
943 2 }
944 2 }
945 2 }
946 2 }
947 2 }
948 2 }
949 2 }
950 2 }
951 2 }
952 2 }
953 2 }
954 2 }
955 2 }
956 2 }
957 2 }
958 2 }
959 2 }
960 2 }
961 2 }
962 2 }
963 2 }
964 2 }
965 2 }
966 2 }
967 2 }
968 2 }
969 2 }
970 2 }
971 2 }
972 2 }
973 2 }
974 2 }
975 2 }
976 2 }
977 2 }
978 2 }
979 2 }
980 2 }
981 2 }
982 2 }
983 2 }
984 2 }
985 2 }
986 2 }
987 2 }
988 2 }
989 2 }
990 2 }
991 2 }
992 2 }
993 2 }
994 2 }
995 2 }
996 2 }
997 2 }
998 2 }
999 2 }
1000 2 }

```

“Nada disso é tudo,  
 Tudo isso é fundamental.”

[H Gessiger, *Engenheiros do Hawaii*]

Christian Becker Bueno de Abreu  
 cbecker@metrosp.com.br