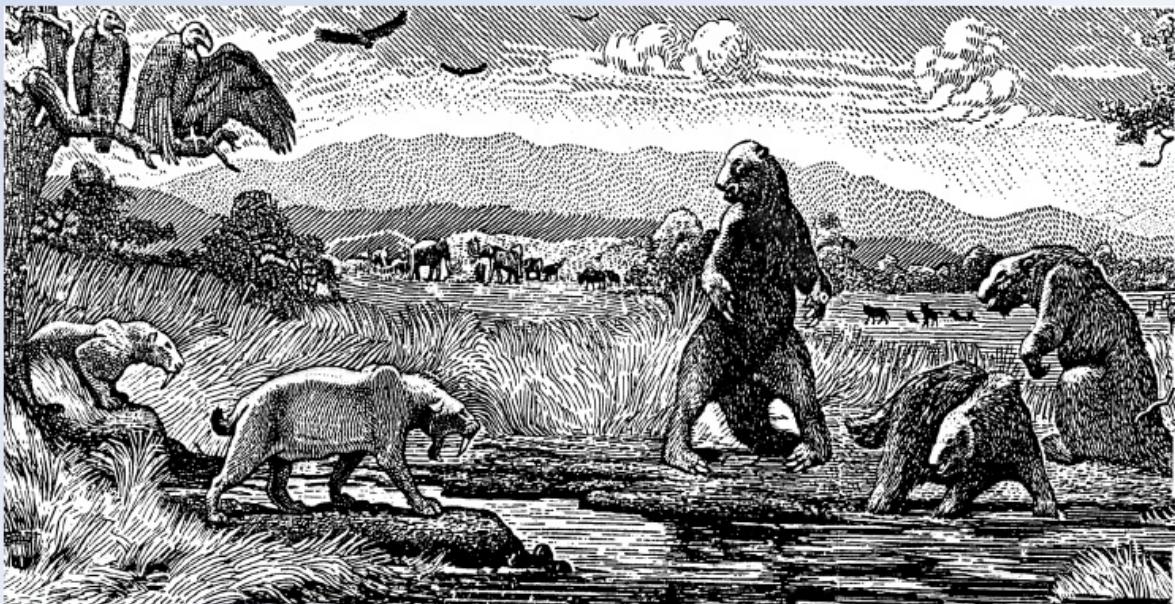


O MÍTICO HOMEM-MÊS

FREDERICK P. BROOKS JR.

O POÇO DE ALCATRÃO



C.R. Knight, Mural dos Poços de Alcatrão de La Brea

The George C. Page Museum of La Brea Discoveries

The Natural History Museum of Los Angeles County

“Um navio na praia é um farol para o mar”

- (explicação breve)

O PRODUTO DA PROGRAMAÇÃO DE SISTEMAS

- Por que não substituir todas as equipes de programação por dedicadas duplas de garagem?
- O que está sendo produzido?

O PRODUTO DA PROGRAMAÇÃO DE SISTEMAS

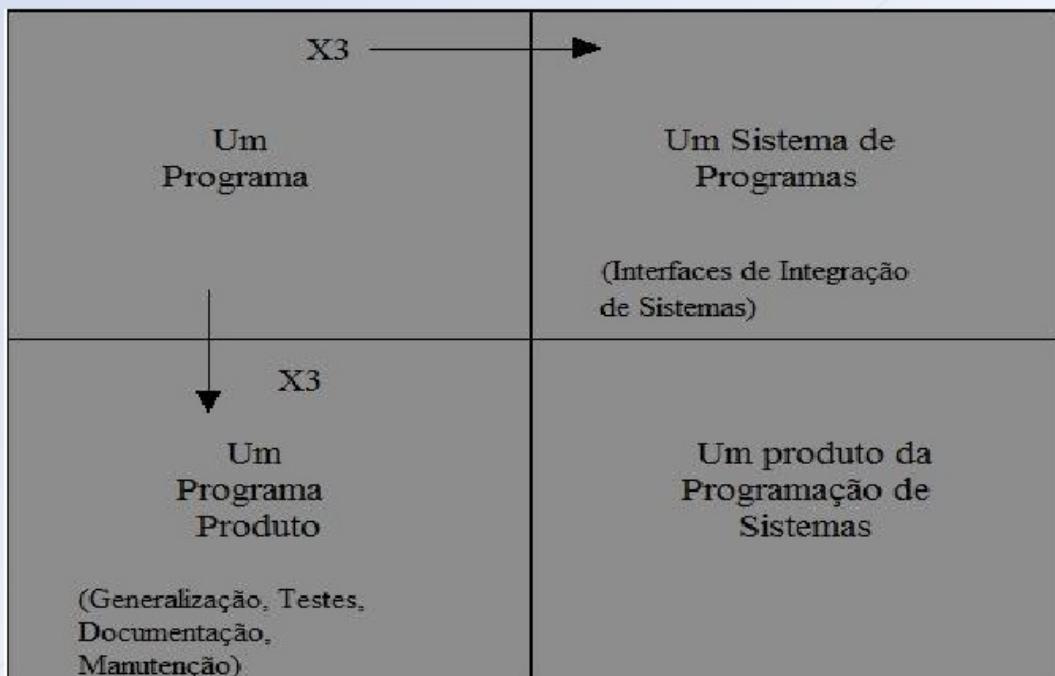


Figura 1.1 Evolução do produto da programação de sistemas

O PRODUTO DA PROGRAMAÇÃO DE SISTEMAS

Programa → Programa Produto
valor aumenta 3x

- Deve ser escrito de forma generalizada;
- Deve ser EFETIVAMENTE testado;
- Deve ser CUIDADOSAMENTE documentado

O PRODUTO DA PROGRAMAÇÃO DE SISTEMAS

Programa



Valor aumenta, no mínimo, 3x

Sistema montado por programas

- Deve ser escrito de forma que as interfaces sejam PRECISAMENTE definidas;
- Deve ser determinado para uma quantidade DETERMINADA de recursos;
- Deve ser testado em CONJUNTO com outros componentes do sistema

O PRODUTO DA PROGRAMAÇÃO DE SISTEMAS

Programa → Produto da programação de sistemas
Valor aumenta 9x

- Diferencia-se do simples programa de TODAS as formas descritas anteriormente.

AS ALEGRIAS DA ARTE

- Por que é divertido programar?
- Que alegrias o praticante dessa arte pode ter como recompensa?

AS ALEGRIAS DA ARTE

- 1^a A satisfação de construir algo;
- 2^a A felicidade de se construir coisas que são úteis para os outros;
- 3^a O fascínio da montagem de objetos complexos;
- 4^a A aprendizagem constante (natureza não repetitiva da tarefa);
- 5^a A delícia de trabalhar em um meio tão maleável.

AS TRISTEZAS DA ARTE

- Nem tudo, porém, é deleite...
- Conhecer as dificuldades intrínsecas ao trabalho ajuda a suportá-las quando elas surgem.

AS TRISTEZAS DA ARTE

- 1^a A execução deve ser perfeita (seres humanos não estão acostumados a serem perfeitos);
- 2^a É raro o programador controlar as circunstâncias de seu trabalho ou mesmo seu objetivo (o programador precisa gastar horas estudando e corrigindo coisas que, em um mundo ideal, deveriam estar prontas, disponíveis e utilizáveis);
- 3^a Há também melancólicas horas de trabalho monótono e cansativo (por exemplo, encontrar um pequeno erro);
- 4^a O produto no qual se trabalhou por tanto tempo parece obsoleto quando fica pronto (ou mesmo antes disso)

AS TRISTEZAS DA ARTE

- (...) a base tecnológica na qual um programador trabalha está sempre evoluindo.
- A obsolência de um uma implementação dever ser medida em relação a outras implementações já existentes e não contra conceitos não implementados.

AS TRISTEZAS DA ARTE

- O que é programar: um poço de alcatrão, no qual muitos esforços, sucumbiram, e uma atividade criativa que traz em si alegrias e tristezas.

O MÍTICO HOMEM-MÊS

- *“Cozinhar bem leva tempo. Se fazemos, você esperar é para servi-lo melhor e deixá-lo satisfeito.”*

MENU DO RESTAURANTE ANTOINE, NOVA ORLEANS

O MÍTICO HOMEM-MÊS

- A maioria dos projetos de software falharam mais por falta de tempo no calendário do que em função da combinação de outras causas.
- Por que isso é tão comum?

O MÍTICO HOMEM-MÊS

- 1º Nossas técnicas para estimativa são muito pouca desenvolvidas;
- 2º Nossas técnicas de estimativa confudem esforço com progresso (homens e meses são intercambiáveis);
- 3º Não temos certeza de nossas estimativas;
- 4º O cronograma de progresso é monitorado de forma precária;
- 5º Quando se admite um atraso no cronograma, a resposta natural é a adição de mais força de trabalho (apagar um incêndio com gasolina).

O MÍTICO HOMEM MÊS

- Consideraremos outros aspectos do problema com mais detalhes:
 - Otimismo;
 - O homem-mês;
 - Testes de Sistema;
 -
 - Estimativa desembasada;
 - Desastre do Cronograma Regenerativo;

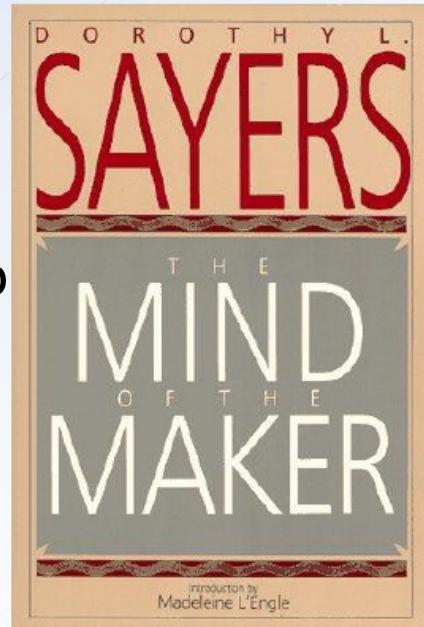
OTIMISMO

- Primeira falsa premissa: “*tudo irá bem...cada tarefa tomará apenas o tempo que ‘deve’ tomar...*”.

A MENTE DO CRIADOR

- Segundo Dorothy L. Sayers:

Atividade criativa:  ideia
implementação
interação



A MENTE DO CRIADOR

- Como criadores humanos de coisas, a incompletude e inconsistência de nossas ideias tornam-se claras apenas durante sua implementação.
- Temos a tendência de culpar o meio físico pela maioria das dificuldades de implementação.

OTIMISMO

- Por se tratar de um meio flexível, se espera encontrar poucas dificuldades de implementação; é então que surge o otimismo.
- Esse otimismo não se justifica, pois, nos esquecemos de que nossas ideias têm FALHAS.

O HOMEM-MÊS

- A segunda falácia se manifesta na unidade de esforço usada em estimativas e cronogramas: o homem-mês;
- O custo varia de acordo com o número de pessoas envolvidas e no tempo disponível. O progresso não;

O HOMEM-MÊS

- (...) o uso do homem-mês como unidade para medir o tamanho de um trabalho é um mito perigoso e enganoso.
- Homens e meses são intercambiáveis apenas quando uma tarefa pode ser dividida entre muitos trabalhadores que **não se
comuniquem entre si**.

O HOMEM-MÊS

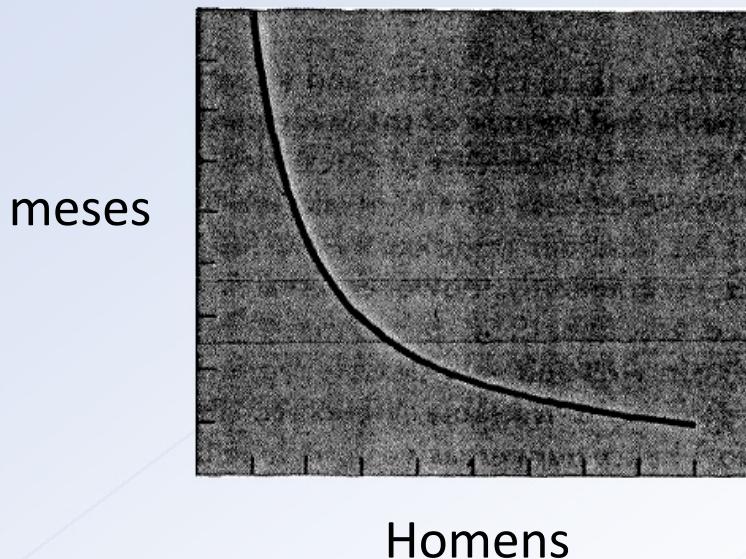


FIGURA 2.1 Tempo *versus* número de trabalhadores em uma tarefa perfeitamente divisível

O HOMEM-MÊS

- Quando uma tarefa não pode ser dividida em função de limitações sequenciais, a aplicação de mais esforço não tem efeito algum no cronograma.
- Ter um filho leva nove meses, independentemente de quantas mulheres sejam responsáveis pela tarefa.

O HOMEM-MÊS

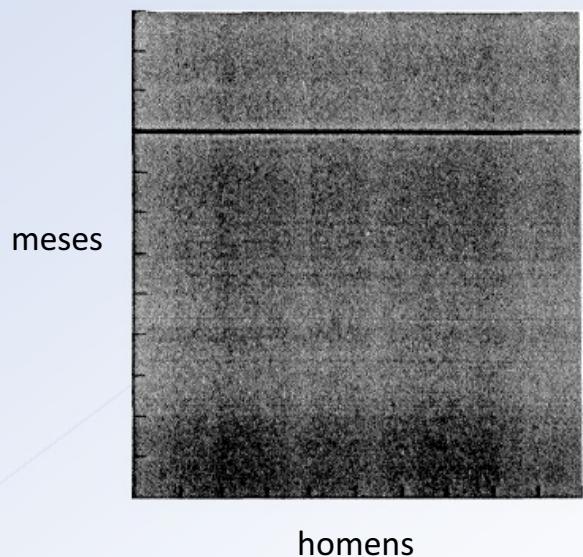


FIGURA 2.2 *Tempo versus número de trabalhadores em uma tarefa indivisível*

O HOMEM-MÊS

- Em tarefas que podem ser divididas, mas que necessitam de comunicação entre as subtarefas, o esforço de comunicação deve ser adicionado à quantidade de trabalho a ser realizado.

O HOMEM-MÊS

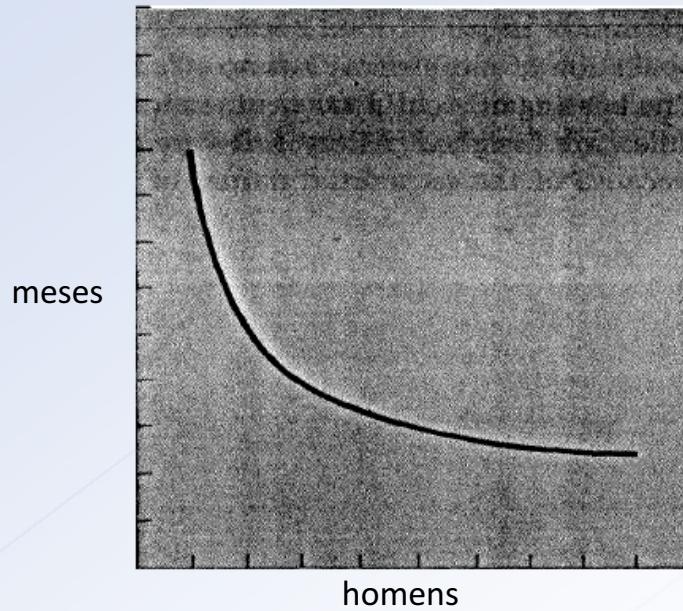
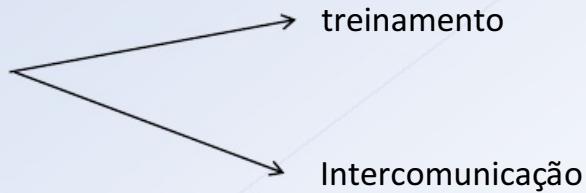


FIGURA 2.3 Tempo *versus* número de trabalhadores em uma tarefa divisível que requer comunicação

O HOMEM-MÊS

- Comunicação



- *Treinamento:* este treinamento não pode ser dividido, assim, parte do esforço adicional varia linearmente com o número de trabalhadores.

O HOMEM-MÊS

- *Intercomunicação:* $n(n-1)/2$
- 3 trabalhadores requerem 3 vezes mais comunicação entre pares do que 2 trabalhadores.
- O esforço adicional de comunicação pode ir totalmente contra a divisão da tarefa scinal e levar à situação da figura 2.4.

O HOMEM-MÊS

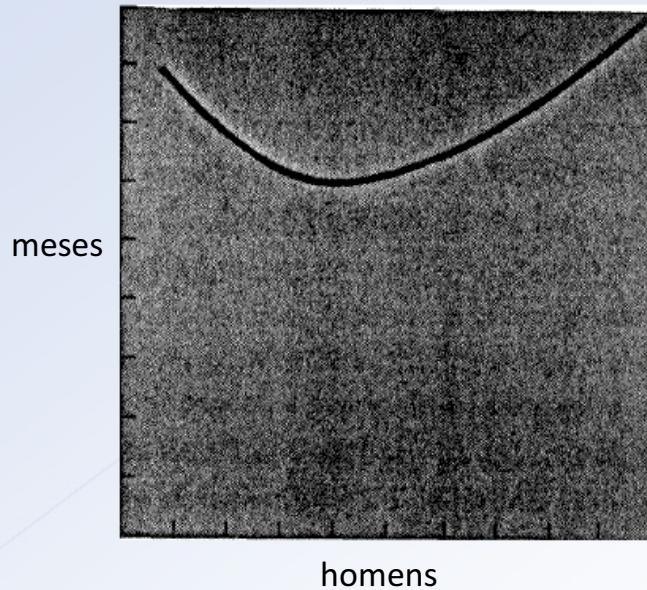


FIGURA 2.4 Tempo *versus* número de trabalhadores em uma tarefa com inter-relações complexas.

O HOMEM-MÊS

- Como a construção de software é por vezes complexa, um exercício em inter-relações aumenta, e não diminui, o tempo no cronograma.

TESTE DE SISTEMA

- Juntamente com a depuração, é a parte do cronograma mais afetada pelos limites sequenciais.
- O tempo requerido para essa tarefa deveria ser zero, mas devido ao otimismo, a fase de testes costuma ser a que é pior estimada em uma tarefa de programação.

TESTE DE SISTEMA

- Regra geral para programar uma tarefa de software:
 - 1/3 planejamento;
 - 1/6 codificação;
 - 1/4 testes de componentes e testes iniciais do sistema;
 - 1/4 testes do sistema, todos os componentes disponíveis.

TESTE DE SISTEMA

- A fração dedicada ao planejamento é maior que a normal;
- A *metade* do cronograma dedicada à depuração do código completo é maior que o normal;
- Para a codificação (fácil de estimar), dá-se apenas um sexto do cronograma.

TESTE DE SISTEMA

- (...) poucos deixam metade do cronograma para testes, mas de fato, usam metade do cronograma real para esse propósito;
- Como o atraso acontece no final do cronograma, ninguém está a par do problema até quase a data de entrega do projeto.

TESTE DE SISTEMA

- Neste ponto, o atraso tem repercussões anormais;
- O projeto está com a equipe completa, o custo por dia está em seu máximo, o software deve dar suporte a outros esforços de negócio e os custos secundários de atraso de tais esforços são muito altos.
- Por isso, é muito importante permitir tempo suficiente para os testes do sistema na programação scinal;

ESTIMATIVA DESEMBASADA

- Tanto para o programador quanto para seu chefe, a urgência do cliente pode governar a data da finalização programada para uma tarefa, mas não sua finalização real;
- O agendamento enganoso para atender ao desejo que o cliente tem por uma determinada data é muito mais comum na nossa disciplina do que em qualquer outra engenharia.

ESTIMATIVA DESEMBASADA

- É muito difícil fazer uma defesa vigorosa e plausível de uma estimativa que não é derivada de nenhum método quantitativo e garantida principalmente por palpites de gerentes.
- Precisamos desenvolver e publicar números de produtividade, de incidência de erros, regras para estimativa...

DESASTRE DO CRONOGRAMA REGENERATIVO

- O que fazer quando um projeto de software essencial está atrasado?
- Suponha que uma tarefa é estimada em 12 homens-mês, entregue a 3 homens por 4 meses, e que existem pontos mensuráveis de verificação A, B, C, D, que estão programados para serem feitos ao final de cada mês (figura 2.5)

DESASTRE DO CRONOGRAMA REGENERATIVO

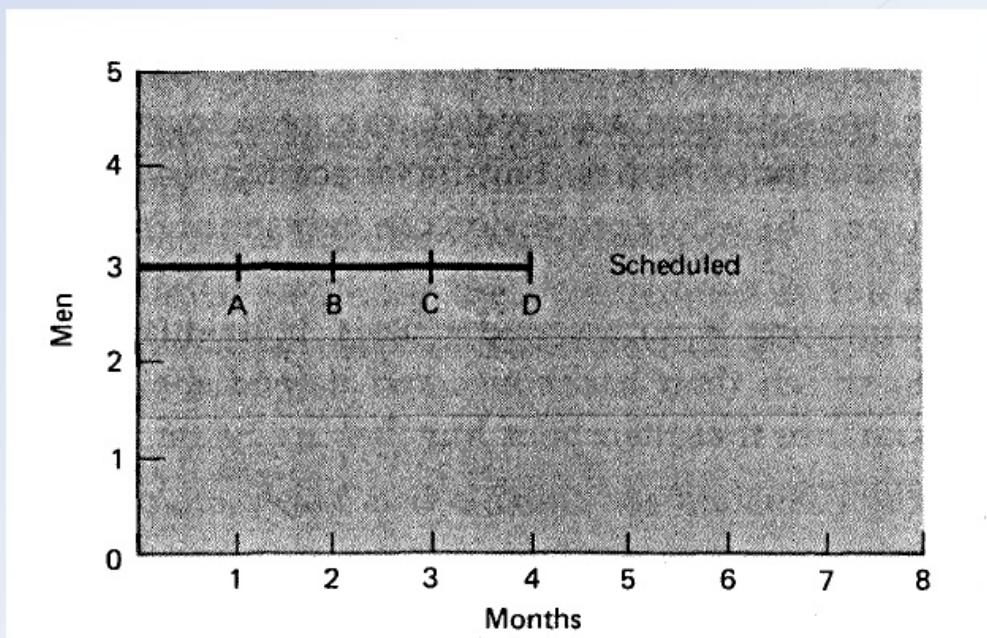


Figura 2.5

DESASTRE DO CRONOGRAMA REGENERATIVO

- Imagine que o 1º ponto de checagem não é atingido até que se passem dois meses.
- Quais são as alternativas que se apresentam ao gerente?

DESASTRE DO CRONOGRAMA REGENERATIVO

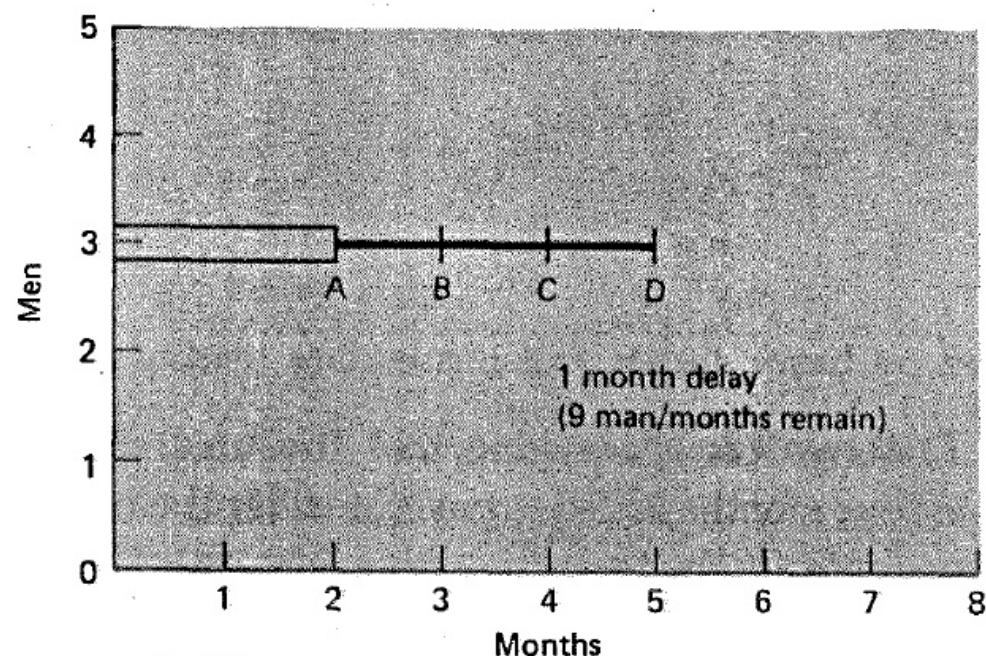


Figura 2.6

DESASTRE DO CRONOGRAMA REGENERATIVO

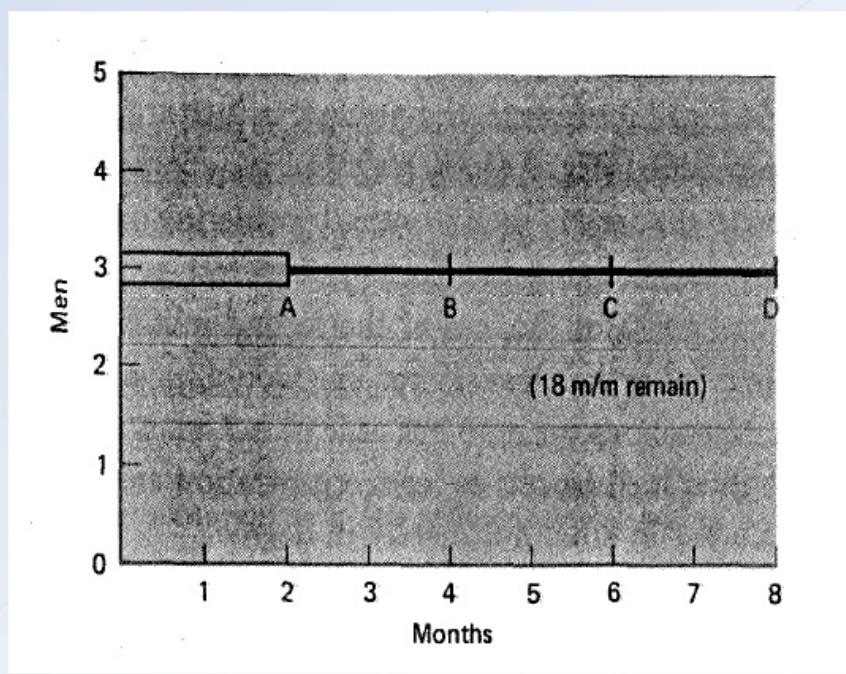


Figura 2.7

DESASTRE DO CRONOGRAMA REGENERATIVO

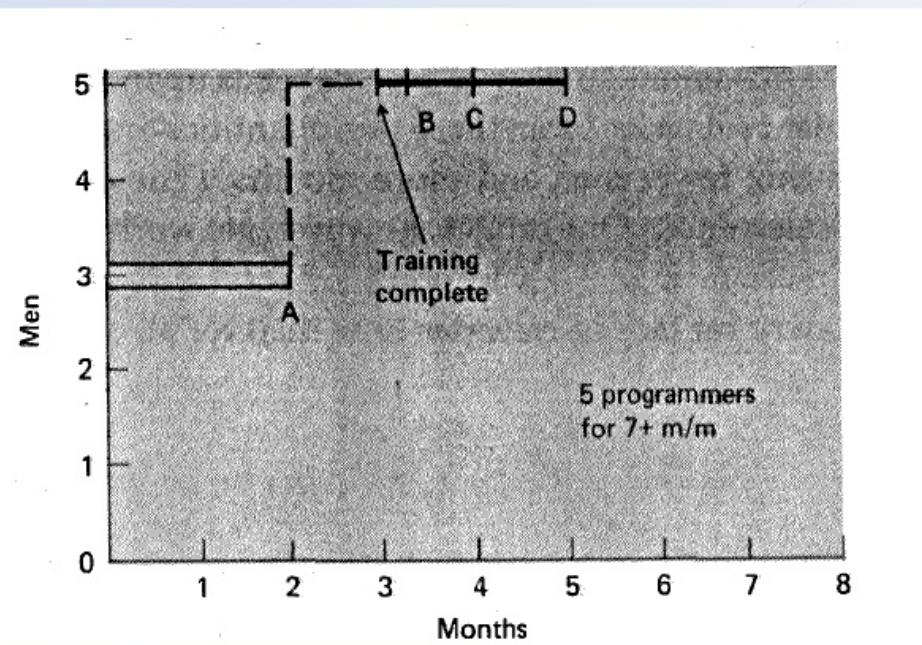


Figura 2.8

DESASTRE DO CRONOGRAMA REGENERATIVO

- Lei de Brooks:

“A adição de recursos humanos a um projeto de software atrasado irá atrasá-lo ainda mais”.

DESASTRE DO CRONOGRAMA REGENERATIVO

- O número de meses de um projeto depende de seus limites sequenciais;
- O número máximo de homens depende do número de subtarefas independentes;
- Não é possível, entretanto, ter cronogramas funcionais utilizando mais homens e menos meses.

A EQUIPE CIRÚRGICA

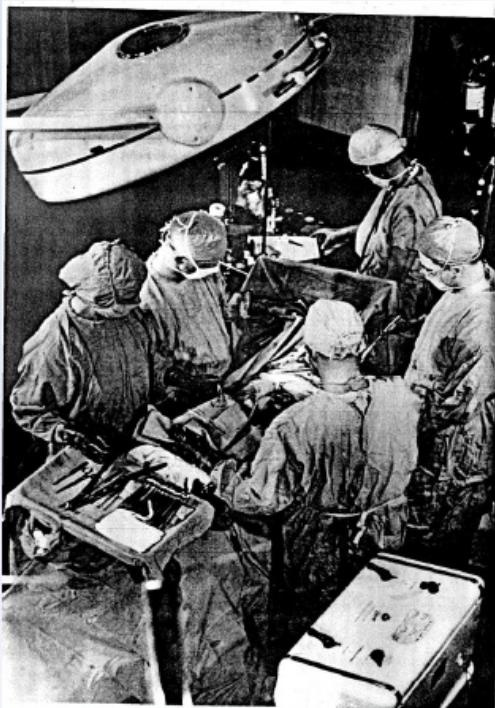


FOTO UPI/O Arquivo Bettman

A EQUIPE CIRÚRGICA

- “Estes estudos revelaram grandes diferenças individuais entre aqueles de alto e baixo desempenho, não raro por uma ordem de grandeza” – SACKMAN, ERIKSON E GRANT

A EQUIPE CIRÚRGICA

- Como construir grandes sistemas em um cronograma significativo?
- Vamos averiguar os dois lado da questão:

O PROBLEMA

- Um estudo de Sackman, Erikson e Grant medindo o desempenho de um grupo de programadores experientes revelou que a variação entre os melhores e o piores desempenhos tinham uma média de 10:1 em medidas de produtividade e 5:1 em velocidade de programação e medidas de espaço.

O PROBLEMA

- A maior parte do custo é a comunicação e a correção dos efeitos da falta de comunicação(depuração do sistema).
- (...)na medida do possível, o melhor é o sistema ser construído por poucas mentes.
- Se um projeto com 200 homens tem 25 gerentes que são os programadores mais experientes e competentes, demita os 175 outros e coloque os gerentes de volta na programação.

O PROBLEMA

- Essa solução falha nos seguintes pontos:
 - A equipe continuaria grande (não deveria exceder 10 pessoas) e precisaria ter ao menos dois níveis de gerência, ou cerca de cinco gerentes;
 - Precisaria de suporte em finanças, pessoal, espaço, secretários e operadores de máquinas;
- Porém, 200 homens não é o suficiente para montar dois desfogadores rotacionais de 360° grandes por

O PROBLEMA

- Eis então o problema do conceito de uma equipe pequena e afiada: *ela é muito lenta para sistemas realmente grandes.*
- Em função da eficiência e da integridade conceitual, é preferível umas poucas e boas mentes trabalhando no projeto e construção;
- Mas para grandes sistemas é preciso uma forma de trabalhar com um número considerável de pessoas para o produto surgir no tempo correto.

A PROPOSTA DE MILLS

- Como as suas necessidades podem ser conciliadas?
- Mills propõe que cada segmento de um grande trabalho seja atacado por uma equipe mas que ela seja organizada como uma equipe cirúrgica.
- Poucas cabeças estão envolvidas no projeto e na construção e, ainda assim, muitas mãos são trazidas para o trabalho.

A PROPOSTA DE MILLS

- O Cirurgião;
- O Copiloto;
- O Administrador;
- O Editor;
- Dois Secretários;
- O Escriturário de programação;
- O Ferramenteiro;
- O Testador;
- O Advogado da Linguagem;

A PROPOSTA DE MILLS

-

Como funciona:

➤ 10 pessoas (7 profissionais), trabalham no problema, mas o sistema é produto de no máximo duas mentes, atuando como apenas uma pessoa.

A PROPOSTA DE MILLS

- As diferenças entre uma equipe de dois programadores e a equipe piloto-copiloto:
 - A divisão do problema;
 - Relação superior-subordinado;
- Todavia, a especialização das funções da equipe é a chave para sua eficiência, já que ela permite um padrão de comunicação radicalmente mais simples entre seus membros.

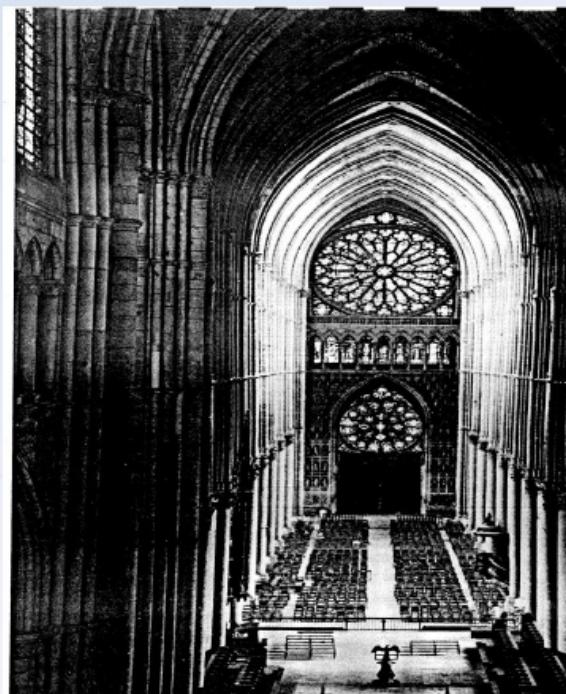
A PROPOSTA DE MILLS

A PROPOSTA DE MILLS

- Escalando:

- ~~Uma grande escala pode ser obtida com poucas interações entre centenas de pessoas estão envolvidas?~~
- O sucesso da escala do processo reside no fato de que a integridade conceitual de cada peça foi radicalmente melhorada;
- Todo o sistema deve ter também integridade conceitual e um arquiteto de sistemas, que deve ~~implementar~~ se ~~apenas~~ nas na

ARISTOCRACIA, DEMOCRÁVIA E O PROJETO DE SISTEMAS



Fotografia de Emmanuel Boudot-Lamotte

ARISTOCRACIA, DEMOCRÁVIA E O PROJETO DE SISTEMAS

INTEGRIDADE CONCEITUAL

- É o ponto mais importante no projeto de sistemas.
- A maioria dos sistemas de programação refletem uma falta de unidade conceitual, decorrente da separação do projeto em muitas tarefas executadas por muitos homens.

INTEGRIDADE CONCEITUAL

- Como atingir a integridade conceitual?
- Esse raciocínio não pressupõe uma elite, ou aristocracia de arquitetos, e uma horda de implementadores plebeus cujos talentos e ideias criativas são suprimidos?
- Como é possível evitar que os arquitetos se distanciem da realidade a ponto de apresentarem especificações caras ou impossíveis de serem implementadas?
- Como garantir que cada simples detalhe de uma especificação seja comunicada de forma clara e precisa ao implementador, de modo que possa ser incorporada ao produto em sua forma exata?

ATINGINDO A INTEGRIDADE CONCEITUAL

- O propósito de um sistema é tornar um computador mais fácil de usar.
- O usuário percebe que é mais fácil especificar qualquer função em particular, mas há muitas mais entre as quais podem escolher e muito mais opções e formatos a serem lembrados.

ATINGINDO A INTEGRIDADE CONCEITUAL

- A facilidade de uso é melhorada apenas se o tempo gasto na especificação das funções exceder o tempo gasto na aprendizagem, memorização e busca em manuais.
- Atualmente, a razão entre o ganho e o custo parece ter diminuído, já que funções cada vez mais complexas têm sido adicionadas.

ATINGINDO A INTEGRIDADE CONCEITUAL

- Nem a funcionalidade ou a simplicidade, isoladamente, definem um bom projeto.
- Ser simples e direto é resultado da integridade conceitual.
- Cada componente deve refletir a mesma filosofia e o mesmo equilíbrio da *desiderata*, o conjunto de aspirações do sistema.

ARISTOCRACIA E DEMOCRACIA

- A integridade conceitual determina que o projeto deve avançar a partir de uma mente, ou de um número muito pequeno de mentes concordantes e consoantes.
- Porém, pressões no cronograma estabelecem que a construção do sistema precisa de muitas mãos.

ARISTOCRACIA E DEMOCRACIA

- Há duas técnicas para resolver esse impasse:
 - Estruturar as equipes de implementação dos programas;
 - Divisão de trabalho entre a arquitetura e a implementação;
- A separação dos esforços de arquitetura e a implementação é uma maneira poderosíssima de obter a integridade conceitual em projetos muito grandes

ARISTOCRACIA E DEMOCRACIA

- Arquitetura de um sistema → especificação completa e detalhada da interface do usuário;
➤ e.g. Manual de programação, manual da linguagem, manual do usuário;
- O arquiteto é o AGENTE DO USUÁRIO;
- Segundo Blaauw: “*Enquanto a arquitetura diz o que acontece, a implementação diz como irá acontecer*”
➤ e.g. Relógio ;

ARISTOCRACIA E DEMOCRACIA

- Não são os arquitetos uma nova aristocracia, uma elite intelectual concebida para dizer ao pôbres e burros implementadores o que fazer?
- Não teríamos um melhor produto se pudéssemos obter boas ideias de todos os integrantes da equipe, seguindo uma filosofia democrática, em vez de restringir a poucos o desenvolvimento de especificações?

ARISTOCRACIA E DEMOCRACIA

- Não apenas os arquitetos têm boas idéias sobre arquitetura, porém a integridade conceitual de um sistema determina sua facilidade de uso.
- Devem existir poucos arquitetos. Se um sistema deve ter integridade conceitual, alguém deve controlar os conceitos.
- Porém, a determinação de especificações externas não é um trabalho mais criativo do que o projeto das implementações; é apenas um trabalho criativo diferente .

ARISTOCRACIA E DEMOCRACIA

- A disciplina é boa para a arte: “*A forma é libertadora*”.
- Da mesma forma, a disposição externa de uma arquitetura valoriza, e não prejudica, o estilo criativo do grupo de implementação.
 - Em um grupo de implementação sem limites, a maior parte do pensamento e debate vai para decisões de arquitetura, enquanto a implementação propriamente dita recebe pouca atenção.

O QUE FAZ O IMPLEMENTADOR ENQUANTO ESPERA?

- Quando se propõe que uma pequena equipe de especificadores realmente passa a escrever sistemas de programação, os implementadores apresentam três objeções:
 - As especificações serão muito ricas em funcionalidade e não irão refletir as considerações práticas de custo;
 - Os arquitetos terão todo o prazer criativo e silenciarão o poder criador dos implementadores;
 - Os muitos implementadores ficarão sem fazer nada, enquanto as especificações passam pelo estreito túnel representado pela equipe de arquitetura;

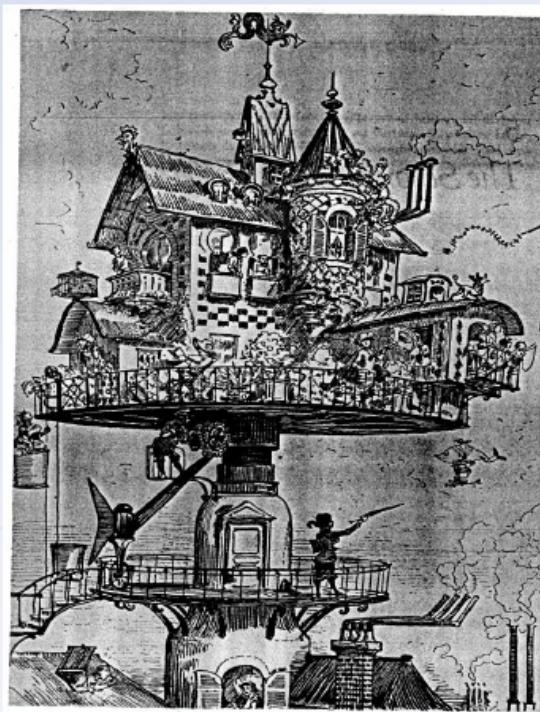
O QUE FAZ O IMPLEMENTADOR ENQUANTO ESPERA?

- 1^a objeção – próximo capítulo;
- 2^a objeção – a implementação também é uma atividade criativa;
- 3^a objeção – não contratar implementadores antes que a especificação esteja completa
 - e.g. Prédio sendo construído

O QUE FAZ O IMPLEMENTADOR ENQUANTO ESPERA?

- A integridade conceitual realmente exige que um sistema reflita uma filosofia única e que a especificação, do ponto de vista do usuário, venha de algumas poucas mentes.

O EFEITO DO SEGUNDO SISTEMA



Casa giratória para tráfego aéreo. Litografia, Paris, 1882

De *Le Vingtième Siècle*, A.Robida

O EFEITO DO SEGUNDO SISTEMA

“Adde parvum parvo magnus acervus erit.”

[Adicione um pouco a pouco e surgirá um grande amontoado.]

OVÍDIO

O EFEITO DO SEGUNDO SISTEMA

- Se separarmos a responsabilidade para a especificação funcional da responsabilidade para construir um produto rápido e barato, qual disciplina limita o entusiasmo criativo do arquiteto?
 - Comunicação meticulosa, cuidadosa e simpática entre arquiteto e construtor e...

DISCIPLINA INTERATIVA PARA O ARQUITETO

- O arquiteto tem duas respostas possíveis quando confrontado com uma estimativa que é muito alta:
 - 1^a: fazer cortes no projeto;
 - 2^a: desafiar a estimativa, sugerindo implementações mais baratas;
- Nessa última, normalmente, o construtor irá retrucar, apresentando sugestões para a arquitetura: não raro, ele estará correto.

AUTODISCIPLINA – O EFEITO DO SEGUNDO SISTEMA

- Durante o primeiro trabalho de um arquiteto, ele sabe que não sabe o que está fazendo, então, ele o faz com muito cuidado e rigor.
- Detalhes de refinamento e requintes de beleza vão lhe surgindo a mente, porém esses elementos são guardados.
- Assim que o sistema é concluído, o arquiteto está pronto para construir um segundo sistema.

AUTODISCIPLINA – O EFEITO DO SEGUNDO SISTEMA

- O segundo sistema é o mais perigoso já projetado por alguém. Suas experiências anteriores irão embasar uma a outra.
- A tendência geral é a de superprojetar o segundo sistema, usando todas as ideias e enfeites que foram cuidadosamente postos de lado no primeiro.
 - “*um grande amontoado*”, segundo Ovídio.
 - e.g. OS/360, TESTRAN, Scheduler

AUTODISCIPLINA – O EFEITO DO SEGUNDO SISTEMA

- Como o arquiteto evita o efeito do segundo sistema?
 - Ele deve estar consciente dos perigos peculiares de tal sistema, exercitando uma autodisciplina fora do comum.
 - Dar a cada pequena função um determinado valor. Esses valores guiarão decisões iniciais e servirão como um guia durante a implementação e um alerta para tudo.

AUTODISCIPLINA – O EFEITO DO SEGUNDO SISTEMA

- Como o gerente de projeto evita o efeito do segundo sistema?
 - Tendo um arquiteto sênior que tenha ao menos dois outros sistemas construídos.
 - Fazer as perguntas certas para garantir quer os conceitos filosóficos e objetivos estejam plenamente refletidos no detalhamento do projeto.

TRANSMITINDO A MENSAGEM



Inclua em Seus Planos o Verbo Descartar



Colapso da ponte de Tacoma Narrows, aerodinamicamente mal projetada, 1940
UPI Photo/Arquivo Bettman

Inclua em Seus Planos o Verbo Descartar

- Engenheiros químicos aprenderam com a experiência que, antes de se levar um experimento bem sucedido do laboratório para a fábrica, deve se ter uma etapa intermediária(planta-piloto).
- Em sistemas de software ,também há o sistema-piloto ,mesmo sabendo que o mesmo será descartado em breve.

O Único Fator Constante é a Própria Mudança

- De acordo com as respostas do cliente em relação ao software ,são feitas as alterações ,pois sem os devidos refinamentos não se chega ao produto final ideal para o cliente.
- Logo descartar faz parte do processo evolutivo do software,e essa ação é constante,ao contrário do software.

Planeje o Sistema para Mudanças

- Todas as versões devem ser devidamente numeradas(versionamento),e documentadas .
- Para que cada mudança seja implementada corretamente ,deve se manter o reuso e o baixo acoplamento.

Planeje a Organização para Mudanças

- Deve ser manter uma organização flexível de todos envolvidos nos projetos, por tanto organizar utilizando o exemplo da equipe cirúrgica.
- Permite que pessoas com mais experiência não deixem de dar boas soluções a problemas no projeto, apenas por pertencer a um cargo mais alto.

Dois Passos Adiante e Um Passo Atrás

- O custo de manutenção de um programa é de 40% ou mais o custo de seu desenvolvimento.
- Quanto mais usuários ,maior a chance de encontrar erros,por tanto maior o custo de manutenção.
- Todas as correções de um software, aumentam as chances de trazer novos problemas.Porque quem conserta não é quem implementa o projeto.

Ferramentas Afiadas



A.Pisano, Lo Scultore, da Capela de Santa Maria del Fiore, Florença, circa 1335
Scala/Art Resource, NY

Ferramentas Afiadas

- As ferramentas utilizadas pelos desenvolvedores tendem a cair em desuso, por conta do surgimento de novas tecnologias.
- E as mesmas ferramentas, utilizadas individualmente, não tem papel decisivo porque o processo de desenvolvimento é criativo e necessita da comunicação entre os componentes da equipe.

Máquinas-alvo

- As máquinas envolvidas nos processos de software são divididas em duas categorias:
- Máquinas-véiculo: dão apoio a construção dos sistemas.
- Máquinas-alvo: onde serão instalados e testados os sistemas.

O Todo e as Partes



O Todo e as Partes

- Sistemas que fazem o impossível ,ou mágica,podem ser programados ,porém não funcionarão.

Projetando sem Bugs

- Integridade conceitual reduz bugs , torna o sistema mais fácil de usar e construir.
- Antes da implementação a especificação deve ser testada ,em busca de completeza clareza.
- Procedimento de Niklaus Wirth,1971,utiliza técnica que começa a apresentar a solução em uma visão macro .E após a cada refinamento ,apresenta uma visão mais micro.

Projetando sem Bugs

- Segundo Bohmm e Jacopini,o uso de GO TO nos sistemas leva a erros lógicos.Ou seja outras estruturas de controle, devem ser usadas.

Depuração de Sistema

- Utilize componentes depurados:os componentes devem ser testados a ponto de encontrar todas as falhas.E mesmo todas a falhas já corrigidas ,erros durante a união dos componentes causará erros.
- Construa degraus suficientes:programas e dados que auxiliam na depuração ,mas não fazem parte do projeto.

Depuração de Sistema

- Controle de mudanças: devem existir cópias distintas do software (versões), e documentos para cada.
- Adicione um componente por vez: a cada componente adicionado, deve se testar o conjunto, e os componentes antigos já adicionados.

Depuração de Sistema

- Quantifique as atualizações: alterações dos componentes por versões atualizadas, devem ser feitas com espaços longos de tempo. Para não causar instabilidade, por conta de um grande número de alterações em um curto espaço de tempo.

Incubando uma Catástrofe



A.Canova,"Ercole e Lica",1802.Hércules leva à morte omensegeiro Licas,que inocentemente lhe entregara a túnica fatal.
Scala/Art Resource,NY

Incubando uma Catástrofe

- O atraso de um projeto pode se dar por causas pequenas ou grandes e desastrosas,a última é facilmente notada e por tanto gerenciável.Mas a primeira,é quase que imperceptível ,pois o atraso é oriundo da soma dos atrasos de cada dia.

Pontos de Checagem ou Pedras no Caminho?

- O cronograma auxilia a manter o projeto em dia,junto com os pontos de checagem.Que de tempos em tempos ,através de algo concreto,pode se saber se os pontos importantes do projeto foram atingidos.
- Já ponto de checagem confuso,é a pedra no caminho,pois não é possível obter uma real idéia se as etapas foram atingidas.

Debaixo do Tapete

- Quando projetos atrasam, os gerentes pode ou não comunicar seus chefes, porém por receio não o fazem. Como chefe duas técnicas são possíveis frente a este problema:
Reduzindo o conflito de papéis e Levantando o Tapete.

Debaixo do Tapete

- Reduzindo o conflito de papéis:o chefe deve dar autonomia ao gerente,para que os problemas sejam resolvidos sem que a autoridade do gerente seja esquecida.
- Levantando o Tapete:em intervalos curtos de tempo ,é necessário um relatório que mostre ao gerente se naquele período o ponto de checagem foi alcançado.

A Outra Face



Stonehenge, Grã-Bretanha

Qual Documentação é Necessária?

- Para usar um programa:
 - 1-Propósito.Razão do programa.
 - 2-Ambiente.Configurações de hardware e sistema operacional em que será executado.
 - 3-Domínio e variações.Tipos de dados válidos e tipos de saída.
 - 4-Funções.O que cada algoritmo faz.
 - 5-Opcões.As possíveis escolhas e respostas das funções.

Qual Documentação é Necessária?

6-Tempo de Execução.Tempo gasto para obter a solução de um certo problema.

7-Precisão e Verificação.Qual precisão pode se esperar das respostas?

Qual Documentação é Necessária?

- Para acreditar em um programa:além da documentação de como funciona um programa,é necessário casos de teste para garantir ao usuário que o sistema funciona.
 - 1-Casos principais,testam as principais funções.
 - 2-Casos de legitimidade,testes com diferentes tipos de dados válidos ,dentro dos limites.
 - 3-Casos de ilegitimidade,testes com tipos de dados inválidos .

Qual Documentação é Necessária?

- Para modificar um programa:é necessário uma boa quantidade de informação ,pois pode ser que a pessoa que implementou não será a mesma que modificará,ou que depois de muito tempo esqueça o papel de cada função.

Qual Documentação é Necessária?

- Diagrama de Fluxos: mostram as decisões possíveis com base nas estruturas, deve ser enxuto para facilitar o entendimento. Diagramas muito poluídos dificultam o entendimento, perdendo o propósito da documentação.

Qual Documentação é Necessária?

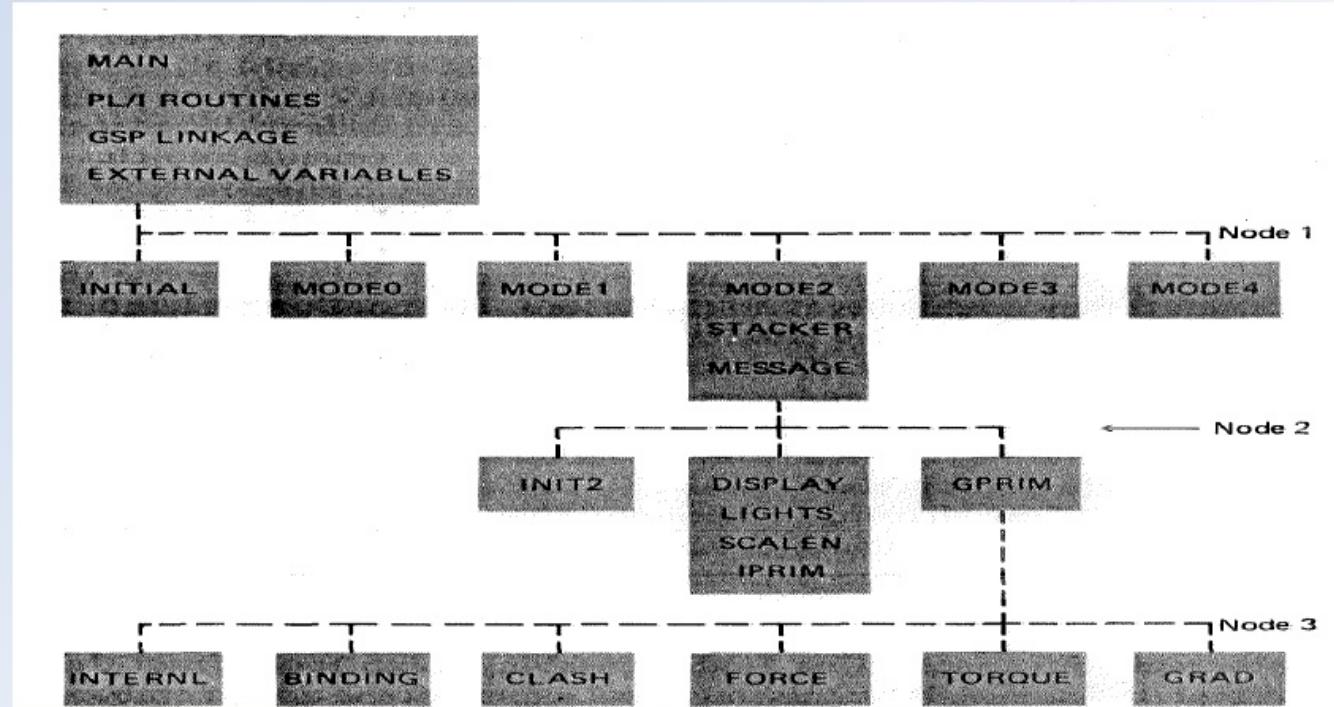
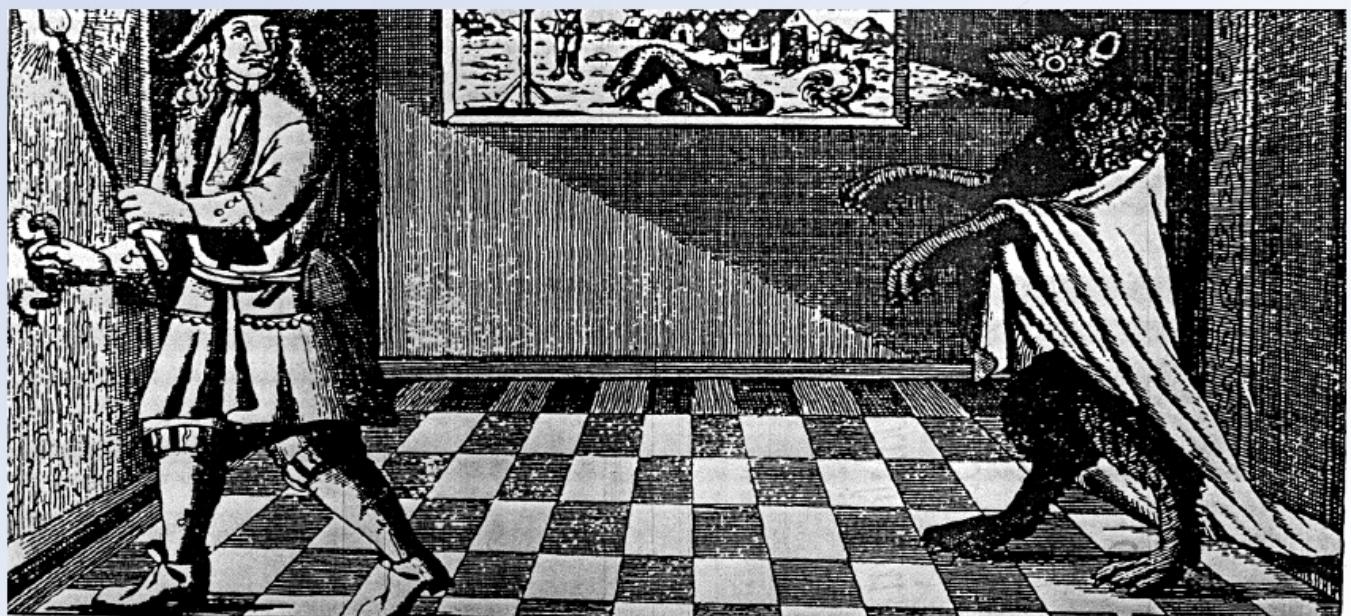


Figura 15.1 O gráfico de estrutura de um programa(Cortesia de W.V.Wright)

Qual Documentação é Necessária?

- Programas Autodocumentados: após a alteração do software ,sua documentação pertinente não é alterada como deveria,pois ambos não estão unificados.Uma solução seria utilizar autodocumentação,que consiste em utilizar comentários nas linhas de código,utilizar nomes de partes do programa que façam menção ao verdadeiro papel da função e indentar o código do programas.

Não Existe Bala de Prata-Essência e Acidente em Engenharia de Software



O Lobisomem de Eschenbach,Alemanha:litografia, 1685.
Cortesia de The Grainger Collection,Nova York.

Não Existe Bala de Prata-Essência e Acidente em Engenharia de Software

- Projetos são comparados com Ibisomens, pois algo familiar pode tornar-se um monstro para o gerente, problemas com orçamento, cronograma e falhas no sistema. E não existe algo que resolva de forma miraculosa!

Não Existe Bala de Prata-Essência e Acidente em Engenharia de Software

- Essência:dificuldades inerentes a natureza do software.
- Acidente:dificuldades na produção ,mas não são inerentes.

Não Existe Bala de Prata-Essência e Acidente em Engenharia de Software

- Complexidade:é uma propriedade essencial que gera dificuldade de entendimento e deficiências no produto.
- Conformidade:a complexidade é adicionada aos projetos sem seguir algum padrão para isso ,por tanto não há conformidade para sistemas complexos.

Não Existe Bala de Prata-Essência e Acidente em Engenharia de Software

- Mutabilidade: softwares estão sempre expostos a mudanças, tanto por culpa do usuário como por surgimento de novas tecnologias ou atualização das mesmas.
- Invisibilidade: por ser algo abstrato, naturalmente o software é algo difícil de se representar.

Não Existe Bala de Prata-Essência e Acidente em Engenharia de Software

- Linguagens de alto nível: o uso de linguagens em que o programador não necessita operar em um nível mais baixo de programação, elimina parte da complexidade.
- Ambientes de programação unificados: diminuem a complexidade , por contar com grande número de ferramentas.

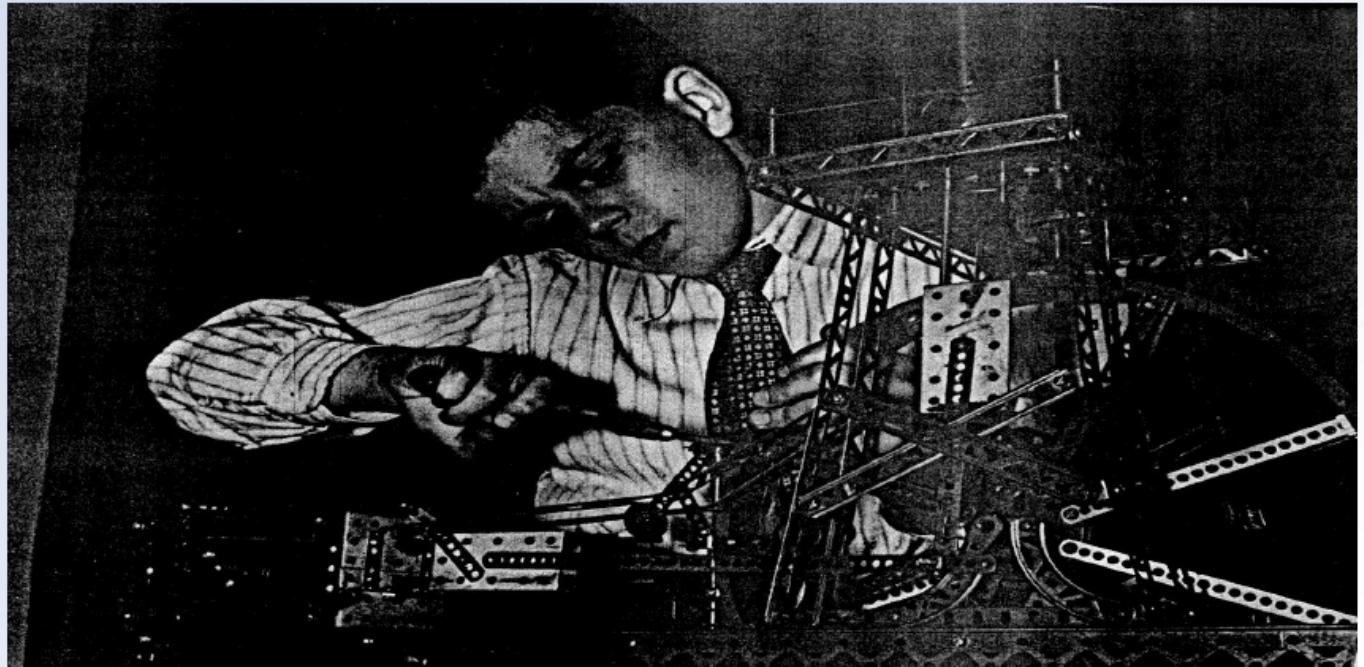
Não Existe Bala de Prata-Essência e Acidente em Engenharia de Software

- Programação Orientada a Objetos:é uma grande aliada na solução da complexidade,porém não é a bala de prata.
- Inteligência Artificial:ganhos na produtividade e qualidade do software ,mas não é uma solução definitiva.

Não Existe Bala de Prata-Essência e Acidente em Engenharia de Software

- Excelentes projetistas: desenvolvimento de software necessita de criatividade, por tanto ,métodos de desenvolvimento nem sempre são úteis. Desenvolvimento feito de modo criativo ,produzem com menor esforço e de forma mais rápida,diminuindo a complexidade.

“Não Existe Bala de Prata”-Mais um Tiro



Montando uma estrutura a partir de peças pré-fabricadas, 1945

Arquivo Bettman

“Não Existe Bala de Prata”-Mais um Tiro

- Neste capítulo,o autor defende seu artigo (capítulo 16) de argumentos contra sua opinião e apóia alguns que são a favor do ponto central do artigo.Que cita a impossibilidade de algo que possa tornar-se a bala de prata(solução definitiva) contra o lobisomem(problemas no projeto).