DESENVOLVIMENTO DE SISTEMAS EMBEBIDOS

(MESTRADO EM INFORMÁTICA)

- SESSÃO 3: Projecto de Software -

JOÃO MIGUEL FERNANDES
Email: miguel@di.uminho.pt
URL: http://www.di.uminho.pt/~miguel

Sumário

1. **Enquadramento**
2. **Mitologia do Software**
3. **Qualidade no Software**
4. **Gestão de Projectos**

1. Enquadramento (1/2)

- **Objectivos deste módulo**
  - **Apresentar os mitos mais comuns associados ao software.**
  - **Identificar os aspectos mais importantes que o software deve ter, para se conseguir um produto de qualidade.**
  - **Discutir alguns problemas associados à gestão de projectos.**
- **Audiência alvo**
  - **licenciados (com ou sem formação na área das TSI) com responsabilidades e experiência comprovada (desejável!) na análise, concepção e implementação de sistemas baseados em software**

1. Enquadramento (2/2)

- **Bibliografia recomendada**
  - **Brooks F.P. (1995). "*The Mythical Man-Month*". Addison Wesley. ISBN 0-201-83595-9.**
  - **Ghezzi C., Jazayeri M., Mandrioli D. (1991). "*Fundamentals of Software Engineering*". Prentice-Hall. ISBN 0-13-818204-3.**
  - **Pressman R.S. (1997). "*Software Engineering: A Practitioner's Approach*". McGraw-Hill, 4th ed. ISBN 0-07-052182-4.**

2. Mitologia do Software (1/15)

- **Mitologia do software**
  - **Muitas das causas subjacentes às incorrecções de software decorrem da existência de uma mitologia sobre o software que foi sendo "desenvolvida" ao longo das décadas e desde o surgimento do primeiro programa de computador.**
- **Mitos do gestor de projecto**
  - **Os gestores de projectos de software, tal como os gestores de outras áreas, estão sujeitos a pressões para manterem os orçamentos sob controlo, para evitarem que os prazos deslizem e para aumentarem os índices de qualidade, provocando a construção mental de vários mitos que lhes aliviam aparentemente a pressão, nem que seja temporariamente.**

2. Mitologia do Software (2/15)

- **Mito #1 do gestor de projecto**
  - **"Já descobri um livro com todos as normas e regras para construir software. Acho que isto é o necessário e o suficiente para que a minha equipa desenvolva correctamente software."**
- **Realidade**
  - **O tal livro pode até, eventualmente, existir mas será efectivamente utilizado?**
  - **Estão os projectistas de software conscientes da sua existência? ou só o gestor de projecto conhece o livro?**
  - **O seu conteúdo reflecte as práticas mais recentes de engenharia de software? é o seu conteúdo completo?**
  - **Na grande maior parte das vezes a resposta a estas questões é "NÃO!"**

2. Mitologia do Software (3/15)

- **Mito #2 do gestor de projecto**
  - **"A minha equipa possui uma ferramenta poderosíssima para desenvolver software. É que eu gastei uma fortuna com os computadores que lhes comprei recentemente".**
- **Realidade**
  - **É preciso muito mais do que o último modelo de computador para realizar um desenvolvimento de software com elevados índices de qualidade.**
  - **As ferramentas de CASE são muito mais importantes do que o hardware para atingir uma boa qualidade e produtividade.**
  - **Apesar disto, a maior parte das equipas de desenvolvimento não utiliza nenhuma ferramenta de CASE.**

2. Mitologia do Software (4/15)

- **Mito #3 do gestor de projecto**
  - **"Se eu vir que o desenvolvimento do software começa a atrasar-se em relação ao previsto, eu contrato, a meio do projecto, mais uma pessoa para integrar a equipa de desenvolvimento".**
- **Realidade**
  - **O desenvolvimento de software não é estritamente mecanicista, tal como uma linha de montagem de automóveis.**
  - **Adicionar pessoas a um projecto de software atrasado provoca ainda mais atrasos.**
  - **Juntar pessoas novas à equipa exige que as outras percam tempo a passar informação às recém chegadas.**
  - **É possível, no entanto, acrescentar pessoas, com resultados positivos, desde essa tarefa seja realizada de forma planeada.**

2. Mitologia do Software (5/15)

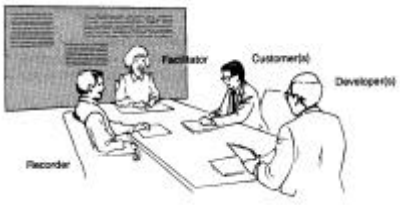
- **Mitos do cliente**
  - **Um cliente que solicita o desenvolvimento de um determinado programa de software pode ser**
    - **o colega da mesa do lado,**
    - **uma outra equipa técnica (como o departamento de venda),**
    - **uma outra empresa que firmou um contrato de aquisição de um novo produto de software.**
  - **Em muitas circunstâncias, o cliente acredita na mitologia do software, porque os gestores de projectos de software e mesmo os elementos da equipa de desenvolvimento não se preocupam em esclarecer os clientes.**
  - **Os mitos levam o cliente a criar falsas expectativas e, em última instância, podem levar à insatisfação para com o fornecedor do produto de software encomendado.**

2. Mitologia do Software (6/15)

- **Mito #1 do cliente**
  - **"Uma descrição genérica daquilo que eu pretendo é mais do que suficiente para que o meu fornecedor comece a bater umas linhas de código. Posso sempre completar a minha ideia mais tarde".**
- **Realidade**
  - **Um incorrecto levantamento de requisitos é a principal causa da má qualidade do software.**
  - **É crucial uma descrição formal e detalhada da informação a manipular, das funcionalidades a disponibilizar, do desempenho a atingir, das constrições impostas, das interfaces, ...**
  - **Só após um diálogo conscientemente executado com o cliente, todos estes requisitos podem ser levantados.**

2. Mitologia do Software (7/15)

- **Mito #1 do cliente**

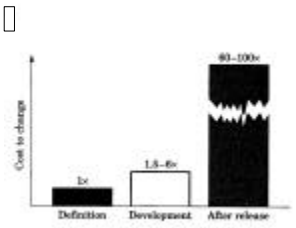


2. Mitologia do Software (8/15)

- **Mito #2 do cliente**
  - **"Os requisitos do produto que encomendei vão mudar ao longo do seu desenvolvimento, mas isso não constitui um problema, porque o produto que contratualizei é software e, como tal, é altamente flexível".**
- **Realidade**
  - **É verdade que os requisitos se alteram com o tempo, no entanto o impacto das mudanças varia com o instante do tempo em que são introduzidas.**
  - **À medida que o tempo de desenvolvimento aumenta, o impacto das alterações de requisitos no custo do produto final aumenta exponencialmente.**

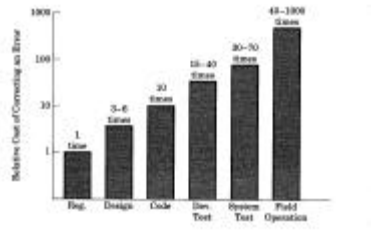
2. Mitologia do Software (9/15)

- **Mito #2 do cliente**



2. Mitologia do Software (10/15)

- **Mito #2 do cliente**



2. Mitologia do Software (11/15)

- **Mitos da equipa de desenvolvimento**
  - **A mitologia do software tem sido um legado das várias gerações de programadores de software que têm construído uma cultura, nem sempre, condizente com a realidade.**
  - **No início da informática, a programação era considerada uma arte e, como tal, devia ser executada por habilidosos com veia , sem regras, nem técnicas bem definidas, mas sim ao sabor do vento dos apetites de quem bate código em cada momento.**
  - **Esta atitude é o principal inimigo da engenharia de software e de qualidade do software.**
  - **Os velhos hábitos ainda se mantém e têm sido muito difíceis de combater.**
  - **Compete às gerações mais novas, com formação formal em engenharia de software, contrariar a prática ainda corrente.**

2. Mitologia do Software (12/15)

- **Mito #1 da equipa de desenvolvimento**
  - **"Depois do código estar todo batido e a funcionar, a nossa tarefa está concluída".**
- **Realidade**
  - **Quanto mais cedo, no fluxo de projecto, se começa a bater código, mais difícil vai ser pôr o programa a funcionar.**
  - **Dados da indústria de produção de software nos E.U.A. indicam que entre 50% a 70% do esforço total despendido no desenvolvimento de um novo produto de software corresponde ao suporte técnico ao cliente após a entrega da primeira versão, para eliminar erros e falhas que a equipa de desenvolvimento não foi capaz de detectar internamente.**

2. Mitologia do Software (13/15)

- **Mito #2 da equipa de desenvolvimento**
  - **"Não é possível realizar testes de qualidade antes de finalizar completamente a implementação da totalidade do código do programa de software".**
- **Realidade**
  - **Um dos mais eficazes métodos de controlo de qualidade no software e que podem ser aplicados desde o início do projecto correspondem à revisão técnica formal do código.**
  - **A execução de uma fase de teste em paralelo com a de desenvolvimento é uma das recomendações básicas para controlar desde o seu início o desenvolvimento de software.**

2. Mitologia do Software (14/15)

- **Mito #3 da equipa de desenvolvimento**
  - **"O único *deliverable* de um projecto de software é o programa de software, propriamente dito, a funcionar".**
- **Realidade**
  - **Um programa de software a funcionar constitui somente um dos *deliverables* do projecto.**
  - **A documentação do software e do próprio projecto constitui uma base sólida para assegurar o sucesso no desenvolvimento de projectos de software, bem como um precioso auxílio para as tarefas de manutenção que se apresentem no futuro.**

2. Mitologia do Software (15/15)

- **Conclusões**
  - **Muitos profissionais de software acabam por reconhecer a falaciosidade destes mitos.**
  - **Infelizmente as práticas correntes continuam a querer fazer com que os mitos sejam reais, mesmo quando a realidade mostra, de uma forma clara e evidente, aos gestores de projecto e às equipas de desenvolvimento, que o caminho a percorrer tem que ser, inevitavelmente, outro; ou seja, a negação dos mitos.**
  - **O reconhecimento da realidade do software é o primeiro passo em direcção a uma correcta formulação de métodos e técnicas efectivamente pragmáticos para o desenvolvimento, com sucesso, de software.**

3. Qualidade no Software (1/18)

- factores críticos -

- **Pessoas**
  - **qualificação profissional**
  - **estabilidade da equipa de desenvolvimento**
- **Tecnologia**
  - **métodos e metodologias utilizados**
  - **ambientes e linguagens de programação**
- **Capacidade de gestão**
  - **gestão dos recursos humanos**
  - **gestão dos projectos (actividades de desenvolvimento)**

3. Qualidade no Software (2/18)

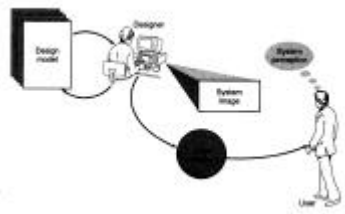
- aspectos de qualidade -

- **Correcção**
  - **Um programa de software diz-se *correcto* se se comportar de acordo com a especificação das funcionalidades que ele deve disponibilizar.**
  - **Esta definição assume que a especificação se encontra disponível e que é possível determinar, sem ambiguidades, se o programa cumpre, ou não, a especificação.**
  - **A correcção é uma propriedade matemática que estabelece uma equivalência entre o software e a sua, suposta, especificação.**
  - **É suposto a especificação ser um modelo daquilo que o cliente pretende do software, embora possa não ser exactamente aquilo que o cliente pretende.**
  - **O software, na melhor das hipóteses, cumpre o que está especificado e nunca aquilo que se pretendia que tivesse sido especificado.**

3. Qualidade no Software (3/18)

- aspectos de qualidade -

- **Correcção: o modelo do sistema**



3. Qualidade no Software (4/18)

- aspectos de qualidade -

- **Fiabilidade**
  - **A *fiabilidade* de um programa depende do seu comportamento estatístico; ou seja, a probabilidade de o programa funcionar tal como esperado durante um intervalo de tempo bem definido.**
  - **A correcção constitui um aspecto de qualidade em termos absoluto, i.e., qualquer desvio relativamente à especificação torna a execução incorrecta, independentemente de quão inofensivo ou fatal possa ser o desvio.**
  - **A fiabilidade é, contrariamente, um aspecto de qualidade em termos relativos, i.e., se a consequência de uma incorrecção de software não for grave, o programa incorrecto manter-se-á fiável.**

3. Qualidade no Software (5/18)

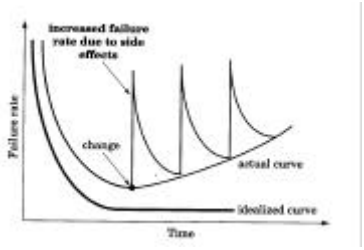
- aspectos de qualidade -

- **Fiabilidade: falhas no hardware**

3. Qualidade no Software (6/18)

- aspectos de qualidade -

- **Fiabilidade: falhas no software**



3. Qualidade no Software (7/18)

- aspectos de qualidade -

- **Robustez**
  - **Um programa diz-se *robusto* se se comportar "razoavelmente", mesmo em circunstâncias que não foram previstas aquando da sua especificação.**
  - **A robustez é, obviamente, um aspecto de qualidade muito difícil de definir, caso contrário seria possível especificar o que se entende por um programa comportar-se "razoavelmente".**
  - **A inclusão de um determinado requisito numa especificação leva um programa a ser classificado relativamente à sua correcção, enquanto que a remoção do mesmo requisito da especificação leva o mesmo programa a ser classificado relativamente à sua robustez, pelo que a robustez e a correcção são dois aspectos de qualidade fortemente relacionados.**

3. Qualidade no Software (8/18)

- aspectos de qualidade -

- **Desempenho**
  - **Em engenharia de software, tal como em outras áreas de engenharia, o desempenho é interpretado como sendo *eficiência*.**
  - **Diz-se que um programa de software é eficiente se utilizar os recursos computacionais de uma forma racional (económica)**
  - **O desempenho (eficiência) de um software pode afectar:**
    - **a sua usabilidade, se criar um ambiente pouco rentável e desconfortável ao seu utilizador**
    - **a escalabilidade, se colocar em causa a sua utilização em circunstâncias de escala superior**
  - **A avaliação do desempenho pode ser realizada:**
    - **medindo a complexidade dos algoritmos;**
    - **analisando modelos do software;**
    - **simulando o comportamento do software.**

3. Qualidade no Software (9/18)

- aspectos de qualidade -

- **Facilidade de utilização**
  - **Um software é fácil de utilizar se os seus utilizadores (clientes) humanos assim o considerarem.**
  - **A facilidade de utilização é entendida de diferentes modos conforme seja um recém utilizador a efectuar a avaliação, ou um utilizador já experiente com o programa em causa.**
  - **A interface gráfica, bem como a sua coerência constituem, frequentemente, os componentes mais importantes da avaliação da facilidade de utilização.**
  - **A correcção e o desempenho também afectam a facilidade de utilização.**
  - **Os outros factores adicionais relevantes para a facilidade de utilização do software são tipicamente de natureza humana: gostos, estados de espírito, classes de utilizadores, etc.**

3. Qualidade no Software (10/18)

- aspectos de qualidade -

- **Facilidade de verificação**
  - **Um software é *verificável* se as suas propriedades forem facilmente verificáveis.**
  - **A correcção e o desempenho são duas propriedades que se pretende sempre que sejam verificáveis.**
  - **A modularização do software, o recurso a práticas disciplinadas na escrita de programas, bem como a utilização de uma linguagem de programação adequada contribuem para a verificabilidade do software.**
  - **Apesar da verificabilidade do software reflectir propriedades internas, em certas circunstâncias, pode ser utilizada para inferir da qualidade externa do programa (ex. o *kernel* de *life-critical solutions*).**

3. Qualidade no Software (11/18)

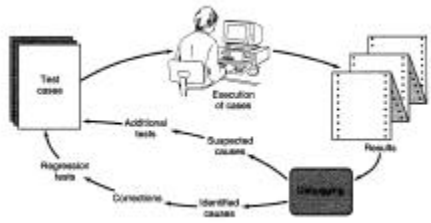
- aspectos de qualidade -

- **Facilidade de manutenção**
  - **O termo *manutenção* de software é utilizado em relação às modificações introduzidas num software após a sua 1ª versão.**
  - **Originalmente, o termo manutenção era utilizado para designar as actividades de correcção de erros (*debug*).**
  - **Actualmente é mais correcto aplicá-lo para referir a melhoria e evolução do software de forma a dotá-lo com funcionalidades não previstas inicialmente.**
  - **A expressão mais correcta e adequada deveria ser *evolução* do software e não manutenção.**
  - **As actividades de manutenção podem ser, genericamente, divididas em:**
    - **correctivas (remoção de erros detectados no software);**
    - **adaptativas (adaptação do software às mudanças do ambiente);**
    - **preditivas (optimização de funcionalidades prevendo mudanças).**

3. Qualidade no Software (12/18)

- aspectos de qualidade -

- **Facilidade de manutenção: *debugging***



3. Qualidade no Software (13/18)

- aspectos de qualidade -

- **Facilidade de reparação**
  - **Um software é *reparável* se permitir que a correcção de defeitos possa ser executada recorrendo a uma quantidade finita de trabalho (esforço técnico humano).**
  - **Um produto de software é muito mais facilmente analisável e reparável se for construído de uma forma modular, do que se for desenhado segundo uma abordagem monolítica ("tijolaço").**
  - **O mero aumento do número de módulos (modularização) presente no programa não resolve o problema da facilidade de reparação, se não se tiver em conta o nível de coesão e de acoplamento existente entre os módulos.**
  - **A facilidade de reparação também pode ser controlado com uma adequada escolha da linguagem de programação a utilizar (em *assembly* é mais difícil reparar erros de software).**
  - **A reparabilidade do software afecta a sua fiabilidade.**

3. Qualidade no Software (14/18)

- aspectos de qualidade -

- **Facilidade de evolução**
  - **Os produtos de software são modificados ao longo do tempo, quer para introduzir novas funcionalidades, quer para alterar algumas já existentes.**
  - **A maleabilidade do software estimula a evolução do software de uma forma pouco cuidada, não existindo, tipicamente, estudos sérios e aprofundados antes das alterações introduzidas para indagar da efectiva possibilidade e viabilidade técnica de tal operação de evolução sem afectar drasticamente os índices de fiabilidade e desempenho para o produto.**
  - **As operações de evolução, após efectuadas, também não são, tipicamente, acompanhadas da documentação respectiva, o que leva a que um produto de software seja cada vez mais difícil de evoluir à medida que sobre ele são realizadas operações de evolução *ad-hoc* e não documentadas.**

3. Qualidade no Software (15/18)

- aspectos de qualidade -

- **Reutilização**
  - **A reutilização está relacionada com a facilidade de evolução.**
  - **Na evolução de um produto, este é modificado para dar origem a uma nova versão do mesmo produto.**
  - **Na reutilização de um produto, este é utilizado, com a introdução de algumas alterações, para dar origem a um outro produto distinto do primeiro.**
  - **Em ambas as situações, a facilidade de evolução é determinante para o sucesso das alterações introduzidas.**
  - **Tipicamente, a reutilização é aplicada a componentes do software e não tanto ao programa de software como um todo.**
  - **A facilidade de reutilização diminui drasticamente, se o desenvolvimento do software não for pensado de raiz com essa intenção, recorrendo, por exemplo, a módulos e componentes.**

3. Qualidade no Software (16/18)

- aspectos de qualidade -

- **Portabilidade**
  - **Um programa de software é considerado *portável* se for possível correr em diferentes ambientes, quer se trate de plataformas de hardware diferenciadas (processadores, barramentos), quer se trate de plataformas de software distintas (sistemas operativos, protocolos de comunicações).**
  - **Com a proliferação de diferentes sistemas operativos (windows98, windowsNT, windowsCE, linux, unix, palmOS ...) e suportes à computação (PCs, workstations, PDAs, telemóveis ...) a portabilidade é um aspecto cada vez mais premente na engenharia de software.**

3. Qualidade no Software (17/18)

- aspectos de qualidade -

- **Interoperabilidade**
  - **A interoperabilidade refere-se à capacidade de um programa de software coexistir e cooperar com outro.**
  - **A interoperabilidade beneficia da normalização das interfaces**
  - **A utilização de tecnologias abertas (*open technologies*) possibilita a montagem de soluções de software complexas, à custa da integração de diversos programas implementados independentemente uns dos outros que, depois de interligados, colaboram entre eles na execução cooperativa de tarefas elaboradas e que não estão ao alcance de qualquer um dos programas isoladamente.**

3. Qualidade no Software (18/18)

- aspectos de qualidade -

- **Conclusões**
  - **A enumeração dos aspectos de qualidade no desenvolvimento de software é um acto relevante, mas poder-se-á tornar inconsequente se não existir forma de desambiguar (tornar objectivo) a percepção que deles se tem.**
  - **Uma das formas de tornar objectiva a interpretação dos aspectos de qualidade em situações concretas consiste na medição (quantificação) de índices de qualidade.**
  - **A definição de métricas para índices de qualidade no software não estão, contudo, livres de polémica, uma vez que se para a a fiabilidade e o desempenho a quantificação é relativamente pacífica, já para a facilidade de utilização, ou para a facilidade de evolução, por exemplo, a quantificação é uma tarefa de muito difícil resolução ou consenso.**

4. Gestão de Projectos (1/13)

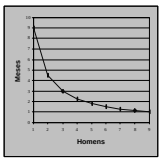
- **Atrasos nos projectos**
  - **Os atrasos são o maior problema para a maioria dos projectos de software.**
  - **As causas para tal realidade são essencialmente as seguintes:**
    - **Técnicas de estimação da duração do projecto pouco evoluídas.**
    - **Atitude optimista perante o projecto.**
    - **Assunção que "homens x tempo" mede o esforço do projecto.**
    - **Progresso do projecto fracamente monitorizado.**
    - **Introdução de mais recursos humanos, quando se percebe que o projecto está atrasado.**
  - **Esta última causa, em vez de acelerar a conclusão do projecto de software, ainda o atrasa mais.**

4. Gestão de Projectos (2/13)

- **Optimismo típico**
  - **Todos os programadores são, por natureza, optimistas.**
    - **"Desta vez, vai tudo correr bem".**
    - **"Encontrei o último bug".**
    - **"O compilador não está bom. O programa não tem erros".**
  - **O 1º erro de gestão é considerar que cada tarefa vai durar aquilo que está previsto de início.**
  - **Uma tarefa tem uma dada probabilidade de correr como previsto.**
  - **Um projecto é composto por diversas tarefas, encadeadas, pelo que é muito baixa a probabilidade de tudo ocorrer sem atrasos.**

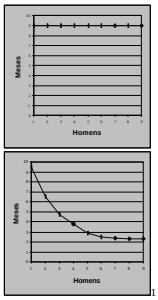
4. Gestão de Projectos (3/13)

- **Homens x Meses**
  - **O custo dum projecto pode medir-se por "homens x meses".**
  - **Porém, tal indicador não deve usar-se para medir o esforço (ou progresso) dum projecto.**
  - **O 2º erro de gestão é considerar que o esforço dum projecto se mede por esse produto.**
  - **Adoptar a métrica "homem x mês" significa que colocar mais recursos humanos reduz o tempo de projecto.**
  - **Os homens e os meses são inter-mutáveis apenas quando uma tarefa pode ser dividida, sem a necessidade de comunicação entre os trabalhadores.**
  - **Ex: apanhar laranjas.**



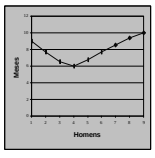
4. Gestão de Projectos (4/13)

- **Homens x Meses**
  - **Quando uma tarefa não pode ser dividida, devido a dependências temporais, o uso de mais homens não afecta o progresso do projecto.**
  - **Ex: o nascimento dum bebé demora 9 meses, qualquer que seja o número de mães considerado.**
  - **Algumas tarefas em software apresentam esta característica, devido à natureza sequencial do *debugging*.**
  - **Em tarefas que podem ser divididas mas que requerem comunicação, este esforço deve ser contemplado.**



4. Gestão de Projectos (5/13)

- **Homens x Meses**
  - **O esforço de comunicação provém de 2 actividades: formação e intercomunicação.**
  - **Cada trabalhador deve receber formação na tecnologia, nos objectivos, na estratégia global e no plano do projecto.**
  - **A formação não pode ser dividida, pelo que o esforço é proporcional ao número de trabalhadores.**
  - **A intercomunicação é mais penalizante.**
  - **O esforço de intercomunicação para n trabalhadores é dado por n(n-1)/2.**
  - **Este esforço pode ser superior ao ganho que a divisão da tarefa proporcionou.**
  - **Os projectos de software seguem este modelo.**



4. Gestão de Projectos (6/13)

- **Teste**
  - **A fase de teste é aquela em que há mais restrições de sequencialidade.**
  - **Assim, o tempo para o teste depende do número e natureza dos erros encontrados.**
  - **Teoricamente, este número devia ser zero! Mas a prática diz-nos que, onde o homem intervém, há erros.**
  - **Devido ao optimismo, o número estimado de erros é normalmente muito menor que aquele que ocorre.**
  - **Por este motivo, a fase de teste é, habitualmente, aquela que é mais mal calendarizada.**

4. Gestão de Projectos (7/13)

- **Teste**
  - **Uma regra (informal) a seguir para a calendarização dum projecto de software de grande complexidade é:**
    - **1/3 para análise e concepção.**
    - **1/6 para implementação (codificação).**
    - **1/4 para teste de componentes e do sistema em fases iniciais.**
    - **1/4 teste de integração (todos os componentes).**
  - **Esta calendarização difere das convencionais no seguinte:**
    - **A parte devotada à análise e à concepção é maior que o habitual.**
    - **A fase de teste corresponde a metade do tempo.**
    - **Metade do tempo de teste é reservado para teste do sistema completo, o que também é maior que o habitual.**
    - **A fase mais fácil de estimar (codificação) corresponde apenas a 1/6 do tempo.**

4. Gestão de Projectos (8/13)

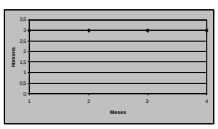
- **Teste**
  - **É normal atribuir-se menos tempo para o teste que aquele que vai ser de facto necessário.**
  - **Não atribuir o tempo suficiente para a fase de teste é particularmente perigoso.**
  - **Uma vez que os atrasos se vão acumulando, as fases finais são aquelas que sofrem as consequências desses atrasos.**
  - **Desta forma, aumenta a pressão sobre a fase de teste:**
    - **Alternativa 1: fazer menos testes (decisão que não garante a qualidade do produto).**
    - **Alternativa 2: aumentar o tempo de projecto (decisão que implica maiores custos e descontentamento do cliente).**
  - **É, pois, extremamente importante atribuir originalmente o tempo necessário e suficiente para a fase de teste.**

4. Gestão de Projectos (9/13)

- **Estimativas**
  - **É típico, em software, aceitar a urgência do cliente, como o tempo necessário para desenvolver um produto.**
  - **Acontece o mesmo noutras áreas? Alguém pede uma vivenda pronta em 2 meses? Ou uma omeleta em 1 minuto?**
  - **Se a omeleta não está pronta passado esse tempo, o cliente tem 2 hipóteses: ou come-a crua ou espera até ela ficar pronta.**
  - **O cozinheiro tem outra alternativa que é aumentar o lume. O resultado é usualmente uma omeleta que tem de ir para o lixo.**
  - **Em software, as estimativas fazem-se com pouco rigor, pois raramente se colectam dados de outros projectos.**
  - **É necessário recolher dados de produtividade, de ocorrência de erros, de temporizações, para poder criar uma base de dados sobre os projectos, para uso em futuras estimações.**

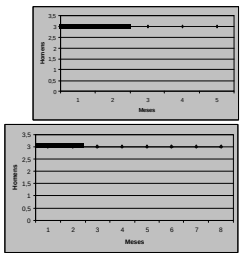
4. Gestão de Projectos (10/13)

- **Atacar os atrasos**
  - **A reacção típica dum gestor dum projecto de software, quando este se atrasa, é recrutar mais recursos humanos.**
  - **Essa decisão pode ser ou não benéfica.**
  - **Suponha-se uma tarefa estimada em 12 h.m, prevista para ser executada por 3h durante 4m. Existem ainda 4 *milestones*, ao fim de cada mês.**
  - **Suponha-se que a 1ª *milestone*, só acontece ao fim de 2 meses.**
  - **Que alternativas deve considerar o gestor?**



4. Gestão de Projectos (11/13)

- **Atacar os atrasos**
  - **A 1ª hipótese é assumir que a tarefa vai fazer-se a tempo e que só a 1ª sub-tarefa foi mal planeada.**
  - **Assim, ainda restam 9 h.m (3 sub-tarefas de 3h.m cada) e 2 meses.**
  - **São pois necessários 4½ homens. Como só temos 3, há que contratar 2 novos colaboradores.**
  - **A 2ª hipótese é assumir que a tarefa vai fazer-se a tempo e que todas as sub-tarefas foram mal estimadas.**
  - **Assim, ainda restam 18 h.m (3 sub-tarefas de 6h.m cada) e 2 meses.**
  - **São precisos 9 homens. Como temos 3, há que contratar mais 6.**



4. Gestão de Projectos (12/13)

- **Atacar os atrasos**
  - **O pressuposto de que a tarefa se pode ainda fazer em 4 meses tem efeitos desastrosos.**
  - **Consideremos a 1ª hipótese.**
  - **Os 2 novos colaboradores, por muito competentes que sejam, tem de ser informados e treinados por um elemento da equipa.**
  - **Se a formação demorar 1 mês, 3 h.m foram usados para trabalho inicialmente não considerado.**
  - **Além disso, a tarefa que inicialmente estava dividida por 3 homens, terá agora que ser distribuída por 5**
  - **Isto implica que algum trabalho já feito seja perdido e que o teste e a integração serão mais longos.**

4. Gestão de Projectos (13/13)

- **Atacar os atrasos**
  - **Portanto, no final do 3° mês, são necessários mais que 7 h.m de esforço e temos 5 homens e um mês.**
    - **7h.m = 12 (total) - 3 (tf A) - 2 (tf B, realizada por 2h no 3° mês)**
  - **Conseguir a tarefa feita em 4 meses, requer então não 2 mas sim 4 novos colaboradores, no fim do 2° mês.**
  - **Passou-se dum projecto com 3 homens para um outro com 7.**
  - **Este processo de adicionar mais pessoas, pode repetir-se se no final do 3° mês, os prazos forem outra vez ultrapassados.**
  - **Sem dúvida que seria preferível manter a equipa com 3 homens e assumir o atraso.**
  - **Podemos então enunciar a lei de Brooks: "Adding manpower to a late software project makes it later".**