Certificar ou não Certificar
Eis a Questão

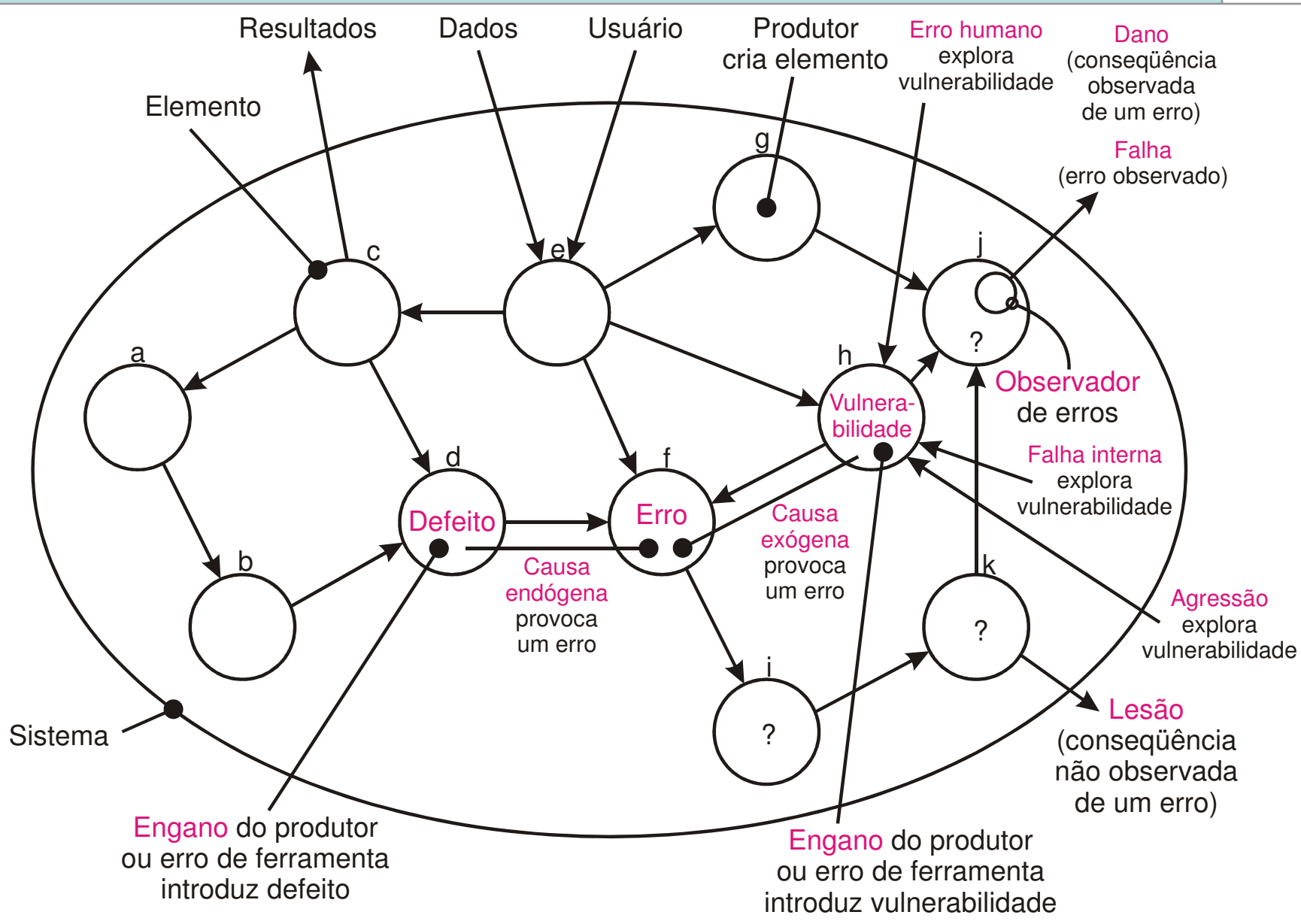
Arndt von Staa

Departamento de Informática

PUC-Rio

Outubro 2010

Visão sistêmica da qualidade



Corretude

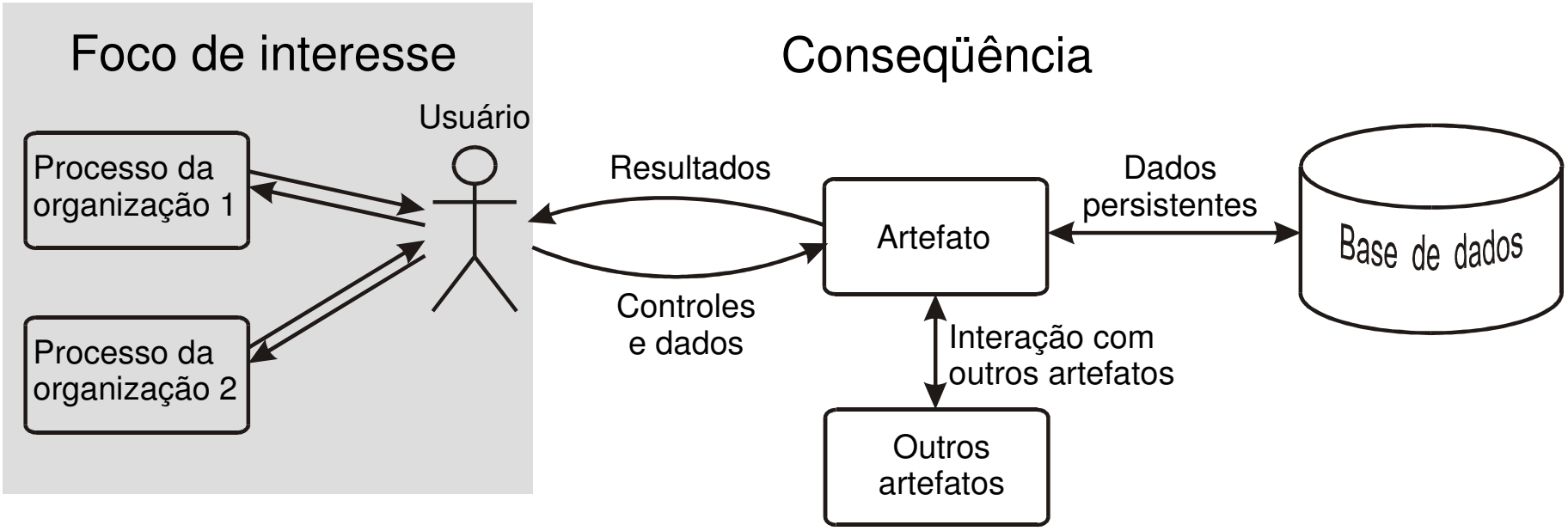
- Artefatos corretos por construção são artefatos isentos de defeitos e vulnerabilidades já antes do primeiro teste
  - artefato satisfaz plenamente o usuário
  - especificação correta e completa
  - artefato consistente com a especificação

ideal ao qual nos devemos aproximar

- Artefatos corretos por desenvolvimento são artefatos isentos de defeitos e vulnerabilidades antes de serem disponibilizados para o usuário
  - artefato pode conter defeito desconhecido
- Artefatos corretos por manutenção são artefatos isentos de defeitos e vulnerabilidades antes de serem redisponibilizados
  - artefato pode conter novo defeito desconhecido adicionado

Qual é o problema?

Erro de foco: o foco não é o sistema de software
mas sim o serviço prestado pelo software ao usuário



O usuário não está interessado em usar software.

O usuário deseja realizar tarefas de forma confiável, pouco aborrecida, pouco cansativa, eficiente (produtiva) com o apoio do software

Qual é o problema?

- Definição equivocada do que entendemos por qualidade
- Definição típica: "Um sistema estará correto se implementar exatamente a sua especificação."
  - e se a especificação não corresponder a o que o usuário precisa, quer ou gostaria de ter?
- Manutenção é realizada também ao desenvolver
  - desenvolvimento incremental
  - evolução das especificações, arquitetura, projetos e códigos
    - incorretos, incompletos ou de baixa qualidade

Qual é o problema?

- ≈ 80% do custo de software deve-se à sua **manutenção**
  - Evolução de sistemas de informação 65%
  - Adaptação a novas leis, regras, plataformas, etc. 18%
  - Correção 17%

Nosek, J.T.; Palvia, P.; "Software Maintenance Management: changes in the last decade"; *Software Maintenance: Research and Practice* 2(3); 1990; pp 157-174

- Custos estimados decorrentes da falta de qualidade (EEUU)
  - Baseado em *surveys* envolvendo desenvolvedores e usuários, o custo anual decorrente de uma infra estrutura inadequada para o **teste** é estimada estar entre US$ 22 (12%) e US$ 59 (33%) bilhões.

NIST; *The Economic Impacts of Inadequate Infrastructure for Software Testing*; Planning Report 02-3; National Institute of Standards & Technology Program Office; Strategic Planning and Economic Analysis Group; May 2002

Qual é o problema?

- Retrabalho inútil é responsável em média por cerca de 50% do custo de desenvolvimento.

Boehm, B.W.; Basili, V.R.; "Software Defect Reduction Top 10 List"; *IEEE Computer* 34(1); Los Alamitos, CA: IEEE Computer Society; 2001; pags 135-137

- São causas de retrabalho inútil
  - especificações inadequadas, erradas, incompletas, ...
  - ferramentas inadequadas
  - processo mal definido
  - indisciplina, incompetência, falta de proficiência, ...
  - desenvolvimento errado
    - inconsistente com a especificação
    - inconsistente com o desejo do usuário
  - manutenção (evolução, correção, adaptação) errada
  - controle da qualidade insuficiente
    - inspeções, verificação estática, medição, testes
  - **...**

Os desafios do desenvolvimento de software

- Desenvolvimento ainda é muito intensivo em pessoal
  - suscetível a falhas humanas
  - variação da competência: 1 para 20, há quem diga 1 para 35
    - competência tende a ser medida só como produtividade
    - competência na realidade envolve:
      - produtividade (funcionalidade correta por unidade de tempo)
      - habilidade em entender o problema a resolver
      - habilidade em dar soluções simples e adequadas
      - habilidade em trabalhar colaborativamente
      - habilidade em transmitir conhecimento
      - disciplina

Boehm, B.W.; *Software Engineering Economics*; Upper Saddle River, NJ: Prentice Hall; 1981

Glass, R.L.; "A Follow-the-Leader Story with a Strange Ending"; *IEEE Software* 22(6); Los Alamitos, CA: IEEE Computer Society; 2005; pags 111-112

Os desafios do desenvolvimento de software

- Quais as alternativas que temos?
  - Reduzir a falibilidade humana através da formação e capacitação de pessoal
    - aumentar a proficiência
    - procurar aproximar o pessoal do nível 35 ☺
  - Reduzir a necessidade de pessoal
    - tornar o desenvolvimento intensivo em capital, exemplos
      - robotizar ➔ geradores, transformadores, analisadores estáticos
      - uso de componentes ➔ compositores, meta-artefatos, arcabouços, bibliotecas, linhas de produto
  - Antecipar e melhorar a eficácia do controle da qualidade
  - Manter sem deteriorar

Desafio: aumentar a proficiência

- Exemplos de sugestões, ensinar e treinar
  - a produzir software com muito menos defeitos
    - disciplina, padrões, boas práticas
  - a empregar sistematicamente técnicas formais leves
    - mostrar como usá-las em aplicações reais
    - não só em *toy problems*
  - a ler software
  - a escrever software para que possa ser lido por outros
  - a trabalhar em grupo (equipe)
  - a corrigir e evoluir
    - segundo alguns manutenção consome cerca de 80% dos recursos de "desenvolvimento"
  - a organizar o trabalho e a dar valor a essa organização
    - planejamento, processos definidos, processos ágeis

Holloway, C.M.; "Why Engineers Should Consider Formal Methods"; Volume 1; *16th Digital Avionics Systems Conference*, 1997; IEEE Computer Society; 1997; pags 16-22

Como e o que ensinar? O SWEBOK realmente ajuda a definir isso?

Desafio: reduzir a dependência de humanos

- Desenvolvimento de ferramentas configuráveis (meta-ferramentas) que apóiem, de forma integrada, equipes multi-disciplinares e distribuídas, realizando ciclos de desenvolvimento, de manutenção e de engenharia reversa em uma variedade de linguagens de programação, projeto, arquitetura e especificação
  - análise estática
  - medição de deficiências (anomalias, *code smells*)
  - transformação automática de representações
  - reflexão automática de representações (engenharia reversa)
  - reengenharia, reorganização da arquitetura (*refactoring*)
  - geração e realização automática de testes
  - ...

Desafio: reduzir a dependência de humanos

- MAS
  - os desenvolvedores devem adorar utilizar as ferramentas disponibilizadas
    - caso contrário viram *shelfware*
  - os desenvolvedores devem receber adequado treinamento e formação
    - um tolo com uma ferramenta continua sendo um tolo
      - a fool with a tool is still a fool
  - as ferramentas devem ser uniformes
    - mesmas ferramentas para um dado domínio de problema e mesmas versões, para toda a equipe
  - o custo / benefício das ferramentas deve ser comprovado experimentalmente – reduzir propaganda e ufanismo
    - idem para processos...
    - custo total de propriedade (TCO - *total cost of ownership*)

Desafio: reduzir a dependência de humanos

- Reutilizar artefatos
  - modelos
  - projetos
  - arquiteturas
  - arcabouços
  - bibliotecas
  - componentes
  - linhas de produto
  - processos
  - padrões
  - . . .

Desafio: reduzir a dependência de humanos

- MAS
  - os artefatos devem
    - ter qualidade comprovada
    - ser fáceis de incorporar a um projeto
    - não induzir enganos (erros humanos) por parte dos desenvolvedores ou mantenedores
    - ser mantidos sem perda de compatibilidade

Thomas, D.; "The Deplorable State of Class Libraries"; *Journal of Object Technology* 1(1); Zürich, CH: ETH Zürich; 2002; pags 21-27

Desafio: controle da qualidade contínuo

- O controle da qualidade deve ser realizado continuamente ao longo de todo o processo de desenvolvimento
  - revisão, inspeção, medição, verificação estática, controle de modelos, testes, instrumentação
  - precisa ser muito mais eficaz
    - ter uma taxa muito maior de detecção de defeitos e de vulnerabilidades
  - precisa ser muito mais eficiente
    - necessitar de muito menos recursos (tempo, labor humano, custo) para realizar o controle
  - o conjunto de artefatos precisa ser capaz de co-evoluir fidedigna e economicamente junto com a evolução ou correção do software

Desafio: controle da qualidade contínuo

- Na realidade o controle da qualidade deveria ajudar a prevenir defeitos
  - especificação e assertivas executáveis usando técnicas formais leves
    - dirige o raciocínio ao desenvolver – redundância de raciocínio
  - desenvolvimento dirigido por contratos
    - *contract driven design*
  - desenvolvimento dirigido por testes
    - *test driven development*
    - *acceptance test driven development*
  - desenvolvimento dirigido por comportamento
    - *behaviour driven development*
    - *feature driven development*
  - desenvolvimento incremental
    - teste real das características à medida que forem desenvolvidas

Desafio: manter sem deteriorar

- Uma parte substancial do esforço é despendido realizando manutenção de software
  - desenvolver um arcabouço de manutenção junto com o desenvolvimento do software
    - entregá-lo junto com o produto
- Recordação:
  - ensinar como manter de forma eficaz
  - ensinar como desenvolver para ser manutenível
    - co-evolução
    - engenharia reversa e reengenharia
    - rejuvenescimento de sistemas legados
      - o sistema entregue hoje, amanhã já é um sistema legado ☹
    - redesenvolvimento sem perda da continuidade do serviço prestado pelo sistema existente
      - sistemas essenciais que precisam mudar de tecnologia

Desafio: manter sem deteriorar

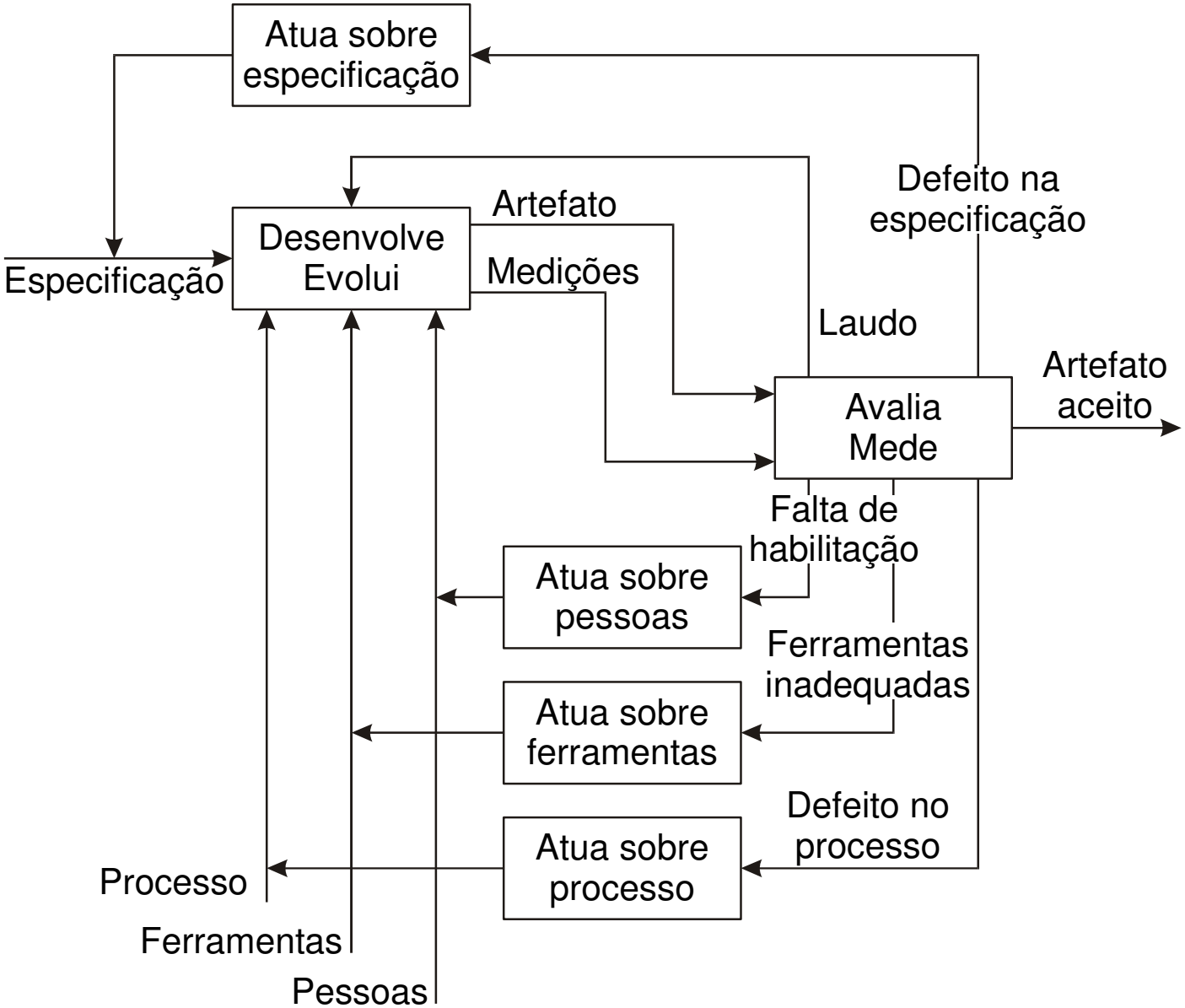
- Visão de software como um ente vivo
  - sistemas de software nascem, crescem, atingem maturidade, evoluem em habilidade (proficiência?), envelhecem, ficam decrépitos e morrem
    - tal como humanos
    - mas podem ser evoluídos tornando-se obsoletos
  - entretanto, enquanto a organização existir, a necessidade do serviço persiste!
  - devem ter vida longa – longevidade
    - independentemente da evolução da tecnologia

Natureza das falhas

Exemplos

- O sistema não resolve, ou resolve mal, o problema do usuário
  - defeito de especificação
- O sistema faz cálculos (tabulações, ...) errados
  - defeito de implementação
- O sistema não está em conformidade com nova legislação
  - defeito por falta de adaptação
- O sistema não prevê erros decorrentes do mal funcionamento do hardware, da plataforma, ...
  - defeito de especificação (requisitos não funcionais)

Melhoria contínua → *feedback*, aprendizado



- Pergunta que espero que estejam se fazendo:

Mas onde entram processos?

Processos: visão bem abstrata

- Processos são sistemas
  - conjuntos de elementos
    - as (boas) práticas
  - conjunto de relacionamentos entre os elementos
    - a organização do trabalho
    - planejamento
  - visando um objetivo comum
    - desenvolvimento e manutenção com elevada produtividade de software satisfatoriamente fidedigno

Pontos positivos de processos "CMM-like"

- Certificação do processo
  - confiança de clientes
  - seleção de fornecedores
- Boas práticas de gerência são desejáveis
  - organizar o trabalho sempre traz benefícios, se inicialmente era desorganizado
- Dispor de um processo padrão comum
  - facilita a comunicação entre organizações
  - facilita o desenvolvimento de ferramentas de apoio à gerência
- Se continuamente mantido e evoluído pode contribuir para uma sensível melhoria de qualidade e produtividade

Orr, K.; "CMM Versus Agile Development: Religious Wars and Software Development"; *Agile Project Management* 3(7); Cutter Consortium; 2002

Pontos negativos de processos "CMM-like"

- Se mal institucionalizado pode engessar
  - tem-se a impressão de que produzir papel (relatórios → evidências) é mais importante do que produzir resultados (funcionalidades operando satisfatoriamente)
- Certificação depende de quem certifica
  - critérios são um tanto fluidos
    - requer certificadores treinados
    - variação individual dos certificadores é inevitável
    - o momento da certificação também introduz variação
  - alguns certificadores mencionam que ajudaram as organizações a melhorarem
    - deveriam meramente identificar conformidades e não conformidades
    - princípio de auditoria: quem faz não controla, quem controla não faz

Pontos negativos de processos "CMM-like"

- Certificação visa principalmente processos e gerência
  - não se preocupa explicitamente com a qualidade dos produtos do ponto de vista do usuário
  - crença (a meu ver errada): seguir um processo bem definido conduz inexoravelmente a sistemas satisfatórios do ponto de vista do usuário
    - especificações não são confiáveis nem estáveis
- Overhead de gerência tende a ser grande
  - gerentes podem tender a ser "chefes" ao invés de "coaches"
  - gerentes podem impor prazos e/ou metas inviáveis
- Manutenção é mencionada raras vezes na literatura de processos (tanto nos baseados em planos como nos ágeis)
  - SPICE menciona
  - o compromisso termina quando o sistema é entregue
    - na prática é aí que começa...

Pontos positivos processos ágeis

- Desenvolvimento como atividade social
  - equipes auto-reguladas e multi-disciplinares
- Foco no produto final adequado ao usuário
  - admitir evolução dos requisitos
  - desenvolvimento incremental
- Teste automatizado
  - critérios de aceitação testáveis passam a ser especificações através de exemplos
  - integração freqüente
  - "refactoring" freqüente
- Interação contínua e explícita com o usuário
  - a aprovação pelo usuário ocorre cedo e passo a passo
  - reduz retrabalho inútil
    - enfatiza as características (*features*) realmente necessárias

Pontos negativos processos ágeis

- Foco excessivo em OO
- Dificuldade de criar e manter equipes (*teams*)
  - dificuldade com o desenvolvimento como atividade social
- Um só usuário residente (*Product Owner*) não necessariamente captura todas as características
- Falta de arquitetura explícita (XP)
- Falta de definição do que seja documentação mínima
- Requer pessoal disciplinado e competente
  - existem suficientes pessoas com essas características?
- Equipes pequenas
  - dá para desenvolver sistemas grandes?
- Manutenção é mencionada raras vezes na literatura de processos ágeis

Reengenharia de "*CMM-like*"

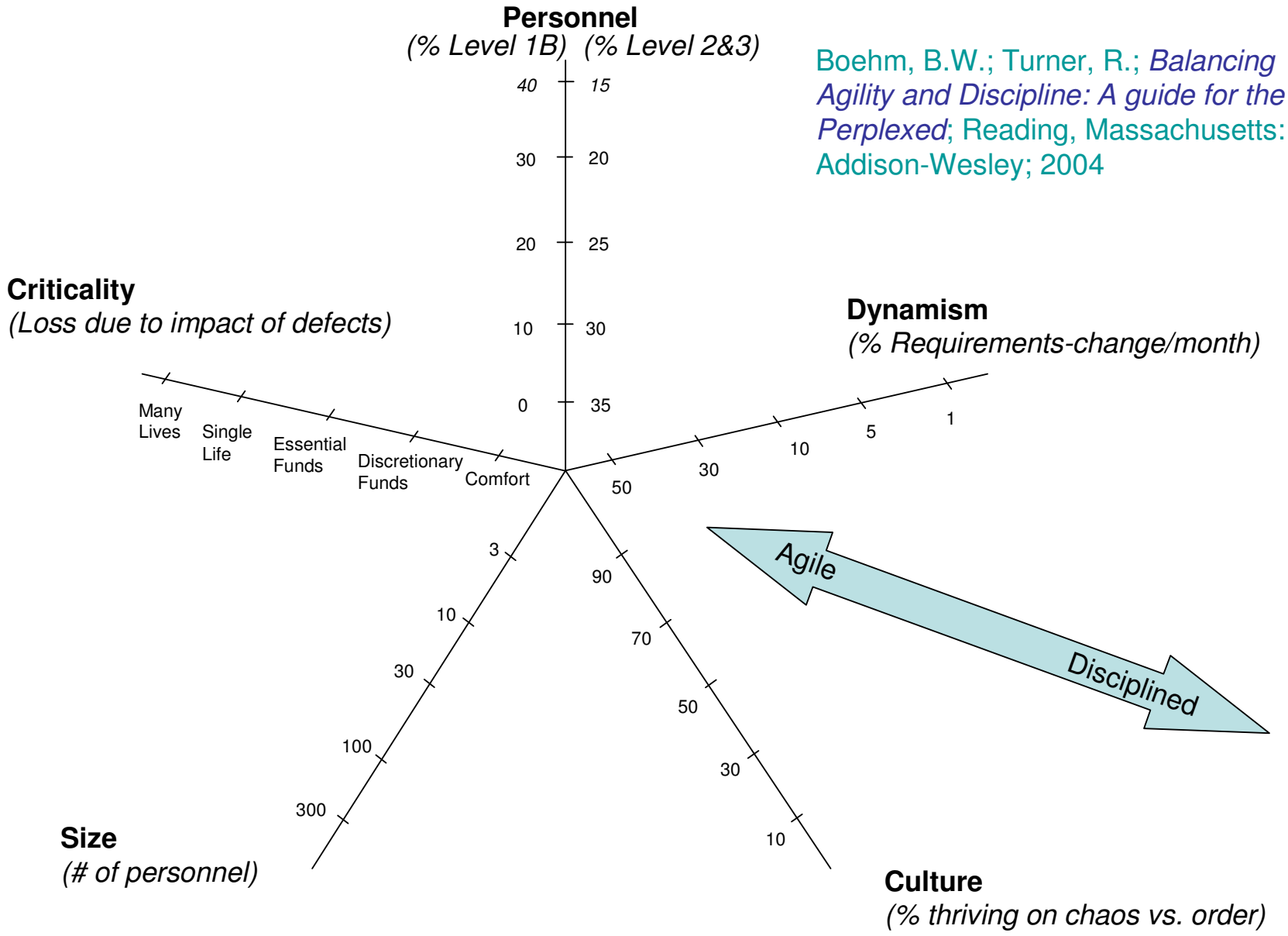
- "Teaching elephants (CMM) to dance"
  - reduzir excesso de documentação pouco útil
    - formulários
    - aprovações (assinaturas)
    - qual seria o mínimo necessário?
      - identificar o que efetivamente contribui e o que é *busy work*
  - enfatizar colaboração
  - incluir ferramentas integradas
    - Huizinga, D.; Kolawa, A.; Automated Defect Prevention: Best Practices in Software Management; Hoboken, New Jersey: John Wiley & Sons; 2007
  - enfatizar o papel de *coach* (treinador) a ser desempenhado pelos gerentes
  - enfatizar resultado e não volume, presença ou similares
  - sair da "cascata"
- http://bradapp.blogspot.com/2006/07/agile-cmmi-and-dancing-elephants.html

Reengenharia de XP

- "Getting XP to work on large projects"
  - prestar mais atenção a
    - especificação
    - arquitetura
    - projeto
    - documentação técnica
    - ferramentas, *frameworks*, geradores
  - enfatizar o papel de coordenador a ser desempenhado pelos gerentes
  - desenvolver técnicas para particionar projetos grandes em vários "adequadamente grandes"
  - desenvolver artefatos de projeto
  - desenvolver e utilizar ferramentas integradas

Lindvall, M.; Muthig, D.; Dagnino, A.; Wallin, C.; Stupperich, M.; Kiefer, D.; May, J.; Kähkönen, T.; "Agile Software Development in Large Organizations"; *IEEE Computer* 37(12); 2004; pp. 26-34

CMMI vs Agile – Dimensions Affecting Selection



Boehm, B.W.; Turner, R.; *Balancing Agility and Discipline: A guide for the Perplexed*; Reading, Massachusetts: Addison-Wesley; 2004

Conclusão

- Procure seguir as diretrizes de algum CMM-*like*
  - o conjunto é consistente e bem elaborado
  - foi amplamente discutido por profissionais experientes
  - consiste em um passo positivo para um o estabelecimento de ambientes de desenvolvimento menos sujeito a crises
  - são excelentes diretrizes para organizar projetos de desenvolvimento
  - não são antagônicos a práticas ágeis
    - muitas práticas ágeis são diretamente aplicáveis, outras são facilmente adaptáveis

Conclusão

- CMM - likes
  - considere seguir o modelo contínuo
  - pode ser implementado em partes
    - identifique o impacto das áreas de processo na organização
      - custo para institucionalizar
      - perda por não institucionalizar
    - ordene do maior impacto positivo para o menor
      - princípio de Pareto
    - institucionalize do maior para o menor
    - reavalie a cada conclusão de etapa de institucionalização
  - podem ser acrescentadas áreas apropriadas para a organização
    - crie áreas de processo levando em conta características da organização e do domínio de problema
    - variedades de CMM-like: SSE-CMM, TMM, MMM, ...
    - ITIL, PMBOK,

Conclusão

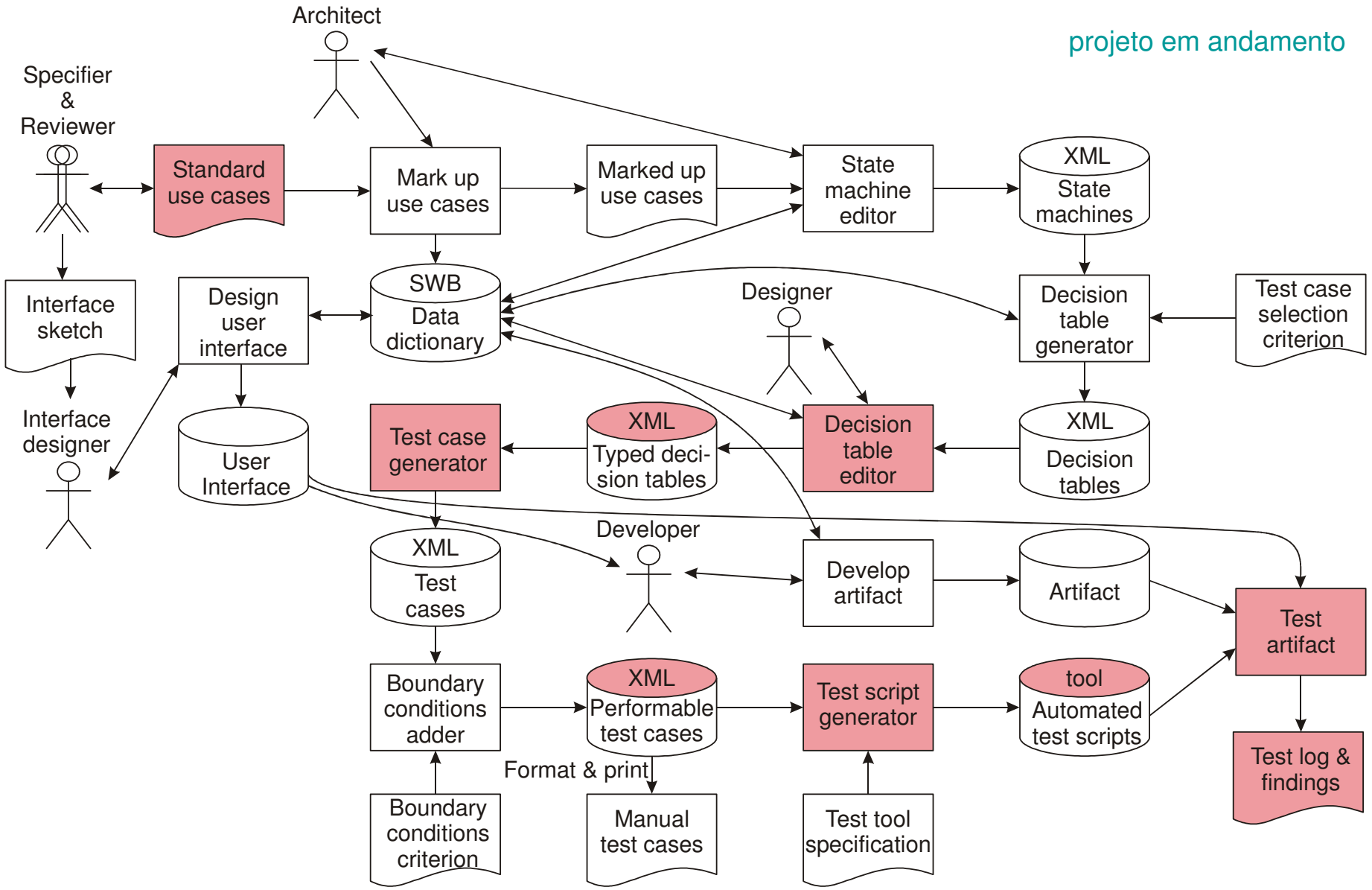
- CMMI e SPICE
  - níveis somente precisam ser implementados completamente se valer a pena obter um certificado
    - considere seguir o modelo contínuo
  - podem ser implementados por área de processo

- MAS, para fazer isso é necessário ter muita experiência, realizar muitos estudos e enfrentar um grande volume de trabalho

Minhas práticas favoritas

- Uso de técnicas formais leves
  - adição de redundância
  - assertivas executáveis
  - verificação automatizada pontual
  - verificação automatizada estrutural
  - sistemas auto-verificantes (*self-checking*)
- Uso de técnicas de verificação estática
  - análise e medição estática de arquiteturas, projetos e código
- Uso de teste automatizado
  - geração dos dados para teste
  - realização dos testes
- Uso de ferramenta de apoio ao projeto (modelagem) e composição do código

Controle da qualidade: testes automatizados



Perguntas?

Arndt von Staa

arndt at inf.puc-rio.br

www.inf.puc-rio.br barra ~arndt