Utilizando o Rational Unified Process para Pequenos Projetos:

Expandindo no eXtreme

Gary Pollice

Rational Software White Paper

TP 183, 3/01

Rational®
the software development company

Índice Analítico

Deleted: 4

Deleted: 4

Deleted: 6

Deleted: 8

Deleted: 9

Deleted: 10

Deleted: 11

Resumo

O produto Rational Unified Process® ou RUP® é uma estrutura de processo de desenvolvimento de software completa fornecida com várias instâncias prontas para uso. Os processos derivados do RUP variam de processos reduzidos—considerando as necessidades de pequenos projetos com ciclos de produtos curtos—a mais abrangentes, considerando as necessidades mais amplas de equipes de projetos maiores e possivelmente distribuídas. Projetos de todos os tipos e tamanhos têm utilizado com êxito o RUP. Esse white paper descreve como aplicar o RUP de maneira reduzida em pequenos projetos. É descrito como aplicar efetivamente técnicas XP (eXtreme Programming) dentro do contexto mais amplo de um projeto completo.

Introdução

Uma Breve História

Em uma manhã, um gerente veio até mim e perguntou se eu poderia passar algumas semanas configurando um sistema de informações simples para um novo empreendimento que estava sendo feito pela empresa. Eu estava entediado com meu projeto atual e me senti entusiasmado com um lançamento, portanto, aceitei o desafio—Eu poderia ser rápido e desenvolver novas soluções excelentes, descarregadas pela burocracia e procedimentos da grande organização na qual eu trabalhava.

As coisas começaram bem. Nos primeiros 6 meses, trabalhei principalmente por iniciativa própria—durante horas e com prazer. Minha produtividade era inacreditável e fiz o melhor trabalho de minha carreira. Os ciclos de desenvolvimento foram rápidos e, normalmente, realizei algumas partes novas e principais do sistema a cada tantas semanas. As interações com os usuários foram simples e diretas—fazíamos todos parte de uma equipe intimamente ligada e poderíamos dispensar formalidades e documentação. Houve também pouca formalidade de design; o código era o design e o design era o código. Tudo correu bem!

Tudo correu bem, por pouco tempo. A medida que o sistema crescia, havia mais trabalho para fazer. O código existente tinha que se desenvolver conforme os problemas mudavam e aprimoramos as noções do que precisávamos fazer. Várias pessoas foram contratadas para ajudar no desenvolvimento. Trabalhamos como uma única unidade, muitas vezes, em pares em partes do problema. A comunicação foi melhorada e a formalidade pôde ser dispensada.

Um ano passou.

Continuamos a agregar pessoas. A equipe cresceu para três, depois para cinco, então para sete. Toda vez que uma nova pessoa iniciava, havia uma curva de aprendizado longa e, se, o benefício de experiência, tornava-se mais difícil entender e explicar o sistema inteiro, mesmo que uma visão geral. Começamos a capturar os diagramas que mostravam a estrutura geral do sistema e os conceitos e interfaces principais mais formalmente.

O teste ainda era utilizado como o veículo principal, para verificar se o sistema realizou o que era preciso. Com muitas pessoas novas no lado do "usuário", descobriu-se que os requisitos informais e os relacionamentos pessoais que funcionavam no início do projeto, não eram mais satisfatórios. Demorou para entender o que esperávamos construir. Por isso, foram mantidos registros gravados de discussões, assim, não era preciso lembrar-se continuamente do que havia sido decidido. Também foi percebido que a descrição dos requisitos e dos cenários de uso, ajudava a educar novos usuários no sistema.

Como o sistema cresceu em tamanho e complexidade, algo de inesperado aconteceu—a arquitetura do sistema requeria atenção. No início, a arquitetura era bastante pensada, também mais tarde em algumas notas rabiscadas ou em gráficos. No entanto, com mais pessoas no projeto, era mais difícil manter a arquitetura sob controle. Como ninguém teve a mesma perspectiva histórica que eu tive, eles não conseguiam ver as implicações de uma mudança específica na arquitetura. Foi preciso definir as restrições arquiteturais no sistema em termos mais precisos. Quaisquer mudanças que podiam ter afetado a arquitetura, requeriam consenso da equipe e, finalmente, minha aprovação. Descobrimos que seria um caminho difícil e houve algumas lições difíceis aprendidas antes que, realmente, foram admitidas por nós mesmos, de que a arquitetura era importante.

Essa é uma história verdadeira. Ela descreve apenas algumas experiências difíceis desse projeto. As experiências são incomuns apenas em um aspecto: muitos de nós estávamos lá do início até o fim, durante anos. Geralmente, as pessoas passam por um projeto e não vêem o impacto que está por vir de suas ações.

Esse projeto poderia ter sido ajudado com um pouco mais de processo. Muitos processos foram obtidos no caminho, mas a falta de processo traz novos riscos. Como a pessoa que investe em ações de alto risco vê apenas altos retornos, grupos que

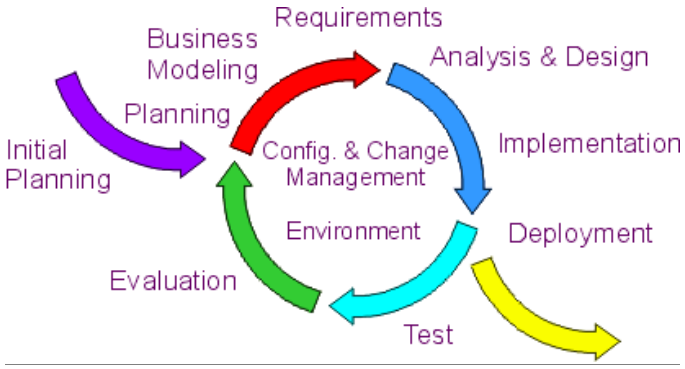
utilizam poucos processos, ignorando riscos chave em seu ambiente de projeto, estão “esperando o melhor, porém, despreparados para o pior.”

Visão Geral

Esse documento discute como aplicar o processo em projetos, como esse agora mesmo descrito. Focalizamos a obtenção do “nível correto” de processo. O entendimento de desafios enfrentados pela equipe de desenvolvimento e pelo ambiente de negócios nos quais são operados, origina o nível correto de formalidade de processo. Quando esses desafios são entendidos, é fornecido apenas o processo suficiente para mitigar os riscos. Não há um único processo adequado para todas as situação, reduzido ou diferente. Nas seções a seguir, é explorada a idéia de que o nível correto de processo é uma função de risco.

Focaliza-se como obter o nível correto de processo, utilizando duas metodologias comuns: o Rational Unified Process ou RUP e Xp (eXtreme Programming). É mostrado como adaptar o RUP em um projeto pequeno e como ele trata de muitas áreas não consideradas pelo XP. A combinação fornece a uma equipe do projeto, a orientação necessária para mitigar os riscos e alcançar sua meta de entrega de um produto de software.

O RUP é uma estrutura e processo desenvolvida pelo Rational Software. É uma metodologia de desenvolvimento iterativa baseada nas seis boas práticas comprovadas de segmento de mercado (consulte o apêndice RUP). Ao longo do tempo, um projeto baseado no RUP passa por quatro fases: Iniciação, Elaboração, Construção e Transição. Cada fase contém uma ou mais iterações. Em cada iteração, você se esforça de várias formas em cada uma das diversas disciplinas, como por exemplo Requisitos, Análise e Design, Teste e assim por diante. O driver chave para o RUP é a *mitigação de riscos*. O RUP foi aprimorado pelo uso em milhares de projetos com milhares de clientes e parceiros do Rational. O diagrama a seguir ilustra o fluxo através de uma iteração típica:



Fluxo de Iteração Típica

Um exemplo de como o risco pode formar um processo, seria possível perguntar se é necessário modelar os negócios. Se houver algum risco significativo no entendimento falho do negócio que fará com que o sistema errado seja construído, provavelmente, será necessário desempenhar algum valor de modelagem de negócios. É preciso ser formal sobre o esforço de modelagem? Isso depende do público-alvo—se uma equipe pequena utilizar o resultado de forma informal, é possível fazer apenas algumas notas informais. Se outros na organização estiverem utilizando os resultados ou revisando-os, provavelmente, será preciso investir algum esforço extra e focalizar mais na correção e no entendimento da apresentação.

É possível customizar o RUP, para ajustar as necessidades de quase todos os projetos. Se nenhum dos processos prontos para uso ou roteiros, se adequar às suas necessidades específicas, você poderá facilmente produzir seu próprio roteiro. Um roteiro descreve como o projeto pretende utilizar o processo e, portanto, representa uma instância de processo específica para esse projeto. O que isso significa é que o RUP pode ser tanto pequeno quanto grande conforme necessário, que é ilustrado neste documento.

XP é um processo centralizado em código reduzido para pequenos projetos (consulte o apêndice XP). É uma invenção de Kent Beck e chamou a atenção do segmento de mercado de software no projeto de folha de pagamento C3 da Chrysler

Corporation em torno de 1997. Como o RUP, é baseado nas iterações que incluem várias práticas, como por exemplo Releases Pequenos, Design Simples, Teste e Integração Contínua. O XP promove várias técnicas efetivadas para os projetos e circunstâncias apropriadas; no entanto, há premissas, atividades e funções ocultas.

O RUP e o XP provêm de diferentes filosofias. O RUP é uma estrutura de componentes, métodos e técnicas de processos que pode ser aplicada a qualquer projeto de software específico; espera-se que o usuário se especialize no RUP. Por outro lado, o XP é um processo mais restrito que precisa de inclusões para ajustá-lo a um projeto de desenvolvimento completo. Essas diferenças explicam a percepção na comunidade de desenvolvimento de software global: as pessoas de sistemas grandes vêem o RUP como a resposta para seus problemas; a comunidade de sistema pequeno vê o XP como a solução para seus problemas. Nossa experiência indica que a maioria dos projetos de software está em algum lugar nesse meio—tentando atingir o nível correto de processo para sua situação. Nem o final do espectro é suficiente para eles.

Quando a distância do RUP é combinada com algumas das técnicas do XP, é atingido o valor correto do processo que recorre a todos os membros de um projeto e trata de todos os riscos principais do projeto. Para uma equipe de projeto pequena que trabalha em um ambiente relativamente de alta confiança, onde o usuário é uma parte integral da equipe, o XP pode funcionar muito bem. A medida que a equipe se torna mais distribuída e o código base cresce, ou a arquitetura não é bem definida, você precisa de algo a mais. É necessário mais do que o XP para os projetos que têm um estilo "contratual" de interação com o usuário. O RUP é uma estrutura a partir da qual você pode estender o XP com um conjunto de técnicas mais robusto, quando necessário.

O restante desse documento descreve um processo pequeno baseado nas quatro fases do RUP. Em cada uma, são identificadas as atividades e os artefatos produzidos.[1] Embora o RUP e o XP identificam funções e responsabilidades diferentes, essas diferenças não são tratadas aqui. Para qualquer organização ou projeto, os membros reais do projeto devem estar associados às funções adequadas no processo.

Início do Projeto — Iniciação

A Iniciação é significativa para novos esforços de desenvolvimento, onde é necessário tratar de riscos de negócios e de requisitos importantes, antes da continuação do projeto. Para projetos que visam melhorias em um sistema existente, a fase de Iniciação é mais rápida, mas ainda se concentra em assegurar que o projeto seja compensatório e que seja possível fazê-lo.

Durante a Iniciação, é criado o caso de negócios para construção do software. A *Visão* é um artefato chave produzido durante a Iniciação. É uma descrição de alto nível do sistema. Ela informa a todos qual é o sistema e também pode informar que o utilizará, por que ele será utilizado, quais recursos devem ser apresentados e quais restrições existem. A *Visão* pode ser breve, talvez apenas um ou dois parágrafos. Muitas vezes, a *Visão* contém os recursos críticos fornecidos pelo software para o cliente.

O exemplo a seguir mostra uma breve *Visão* escrita para o projeto, para reprojetar o Web site externo do Rational.

> *Para conduzir a posição do Rational como o líder mundial no e-development (ferramentas, serviços e boas práticas), melhorando os relacionamentos de clientes por meio de uma presença da Web dinâmica e personalizada com o fornecimento de conteúdo automático, de suporte e direcionado do visitante. O novo processo e a ativação de tecnologias irão autorizar os provedores de conteúdo, a acelerar a publicação e a qualidade do conteúdo através de uma solução simplificada e automatizada.*

As quatro atividades essenciais de Iniciação especificadas no RUP são:

- **Formular o escopo do projeto** — Se um sistema estiver sendo produzido, será preciso saber qual é ele e como irá satisfazer os investidores. Nessa atividade, foram capturados o contexto e os requisitos mais importantes com detalhes o suficiente para derivar os critérios de aceitação do produto.
- **Planejar e preparar o caso de negócios** — Com a *Visão* como um guia, a estratégia de mitigação de riscos é definida, um plano de projeto inicial é desenvolvido e os comércios de custo, planejamento e rentabilidade são identificados.

[1] O XP define três fases: Exploração, Confirmação e Direção. Elas não mapeiam de forma satisfatória as fases do RUP, portanto, optamos por utilizar as quatro fases do RUP para descrever o processo.

- **Sintetizar uma arquitetura candidata** — Se o sistema em consideração tiver pouca novidade e tiver uma arquitetura bem entendida, será possível pular essa etapa. Assim que se sabe o que o cliente requer, é estabelecido o tempo para investigar as arquiteturas candidatas potenciais. A nova tecnologia apresenta a possibilidade de soluções novas e aprimoradas para problemas de software. A demora no processo para avaliar os comércios de compra versus de construção, bem como selecionar tecnologias e talvez desenvolver um protótipo inicial, pode reduzir alguns riscos principais do projeto.
- **Preparar o ambiente de projeto** — Todo projeto precisa de um ambiente de projeto. Se você utilizar algumas das técnicas do XP, como por exemplo, programação em pares ou técnicas mais tradicionais, será preciso determinar os recursos físicos, as ferramentas de software e os procedimentos que a equipe irá seguir.

Não é dispensado muito "tempo do processo" para desempenhar a Iniciação em um projeto pequeno. Muitas vezes, é possível concluir a Iniciação em poucos dias ou menos. As seções a seguir descrevem os artefatos esperados, diferente da *Visão*, dessa fase.

Um Caso de Negócios Aprovado

Os investidores têm a oportunidade de concordar que, a partir de uma perspectiva de negócios, o projeto seja compensatório. O RUP e o XP concordam que é melhor saber antecipadamente se o projeto é compensatório, em vez de gastar recursos valiosos em um projeto perdido. O XP, assim como é descrito em *Planejando o Extreme Programming*[1], é confuso no aspecto em como os projetos estão sendo apresentados e quais funções estão envolvidas (parece mais claro no contexto de um negócio ou sistema existente), mas na fase de Exploração, o XP trata de artefatos de iniciação do RUP equivalentes.

Se você considerar os problemas de negócios informalmente como no XP ou se fizer do caso de negócios um artefato do projeto excelente, tal como com o RUP, será preciso considerá-los.

Lista de Riscos

A Lista de Riscos é mantida por todo o projeto. Pode ser uma lista de riscos simples com estratégias de mitigação planejadas. É dado prioridade aos riscos. Em nenhum momento, ninguém associado ao projeto pode ver quais são os riscos e como você os trata. Kent Beck descreve um conjunto de riscos tratados pelo XP e como ele os trata, mas nenhuma abordagem geral para controle do risco é fornecida.[2]

Plano de Projeto Preliminar

Planos de estimativas de recursos, de escopo e de fase são incluídos neste plano. Em qualquer projeto, essas estimativas mudam continuamente e é necessário monitorá-los.

Plano de Aceitação do Projeto

Se você tiver ou não um plano formal, dependerá do tipo de projeto. É necessário decidir como o cliente avaliará o sucesso do projeto. Em um projeto XP, ele tem o formato de testes de aceitação criados pelo cliente. Em um processo mais geral, o cliente pode realmente não construir os testes, mas os critérios de aceitação devem ser conduzidos pelo cliente diretamente ou através de outra função, como por exemplo o Analista de Sistemas, que interage diretamente com o cliente. Podem existir outros critérios de aceitação, como por exemplo a produção de documentação e de ajuda do usuário final, que não é tratada pelo XP.

---

[1] Este é um dos três manuais atualmente publicado no eXtreme Programming.

[2] No *eXtreme Programming eXplained*, Kent Beck descreve oito riscos e como o XP trata deles (pp. 3-5). O leitor foi convidado a examiná-los e determinar se são suficientes. Acredita-se que existem muitos outros erros e uma estratégia de controle de riscos geral é uma parte necessária de qualquer processo.

Um Plano para a Iteração Inicial Elaboração

Em um projeto baseado no RUP, planeje cada iteração detalhadamente no final da iteração anterior. No final da iteração, avalie o progresso em oposição aos critérios designados no início da iteração. O XP fornece algumas técnicas recomendáveis para monitoramento e medida do sucesso de uma iteração. As métricas são simples e você pode facilmente incorporá-las no plano de iteração e critérios de avaliação.

Elaboração

A meta da fase de Elaboração é criar a linha de base para a arquitetura do sistema, a fim de fornecer uma base estável para a parte principal do esforço de design e de implementação da fase de Construção. A arquitetura se desenvolve a partir de um exame dos requisitos mais significativos (aqueles que têm grande impacto na arquitetura do sistema) e de uma avaliação de risco. A estabilidade da arquitetura é avaliada através de um ou mais protótipos de arquitetura.

No RUP, as atividades de design são focalizadas na noção de arquitetura de sistema e, para sistemas de software intensivo, na arquitetura de software. Utilizar as arquiteturas de componentes é uma das seis boas práticas de desenvolvimento de software incluída no RUP, a qual recomenda levar tempo no desenvolvimento e na manutenção da arquitetura. O tempo gasto neste esforço mitiga os riscos associados a um sistema sensível e inflexível.

O XP substitui a noção de arquitetura por "metáfora." A metáfora captura a parte da arquitetura, enquanto que o restante da arquitetura é desenvolvido como um resultado natural de implementação de código. O XP considera que a arquitetura surge da produção do design mais simples e, continuamente, da recriação do código.

No RUP, a arquitetura é mais do que metáfora. Durante a Elaboração, construa arquiteturas executáveis, a partir das quais é possível reduzir muitos dos riscos associados ao atendimento de requisitos não funcionais, como por exemplo, o desempenho, a confiabilidade e a robustez. Na leitura da literatura do XP, é possível deduzir foi feito esforço em vão em alguns dos itens que o RUP prescreve para Elaboração, especificamente, a concentração indevida na qual o XP chama a infra-estrutura. O XP informa que o esforço empenhado na construção da infra-estrutura antes de sua necessidade, leva a soluções excessivamente complexas e à produção de coisas que não possuem valor para o cliente. No RUP, a arquitetura e a infra-estrutura não são idênticas.

A abordagem para a arquitetura é totalmente diferente entre o RUP e o XP. O RUP aconselha a prestar atenção na arquitetura, para evitar os riscos associados ao escopo aumentado com o passar do tempo, o tamanho adicional do projeto e a adição de novas tecnologias. O XP considera uma arquitetura existente ou que a arquitetura seja simples o suficiente ou muito bem entendida, para que possa ser desenvolvida conforme codificada. Ele aconselha a não planejar para amanhã, mas implementar hoje. A crença é que o amanhã terá autonomia se você manter o design o mais simples possível. O RUP o convida a avaliar os riscos de tal proposta. Se o sistema ou partes dele precisar ser novamente escrito no futuro, o XP informará que ainda será melhor e, muitas vezes, menos caro do que planejar a possibilidade agora. Para alguns sistemas, isso será verdadeiro e, utilizando o RUP, sua consideração de risco durante a fase de Elaboração o levará a essa conclusão. O RUP não garante essa veracidade para todos os sistemas e a experiência sugere que para sistemas maiores, mais complexos e sem precedentes, isso pode ser desastroso.

Enquanto é verdade afirmar que prestar muita atenção às possibilidades futuras que podem não ocorrer nunca pode ser inútil, prestar a quantidade certa de atenção no futuro é uma coisa prudente a ser feita. Quantas empresas podem se permitir, continuamente, reescrever ou mesmo reformular o código?

Em qualquer projeto, é necessário executar pelo menos essas três atividades durante a Elaboração:

- **Definir, validar e criar a linha de base da arquitetura** — Utilize a lista de riscos para desenvolver a arquitetura candidata a partir da fase de Iniciação. Estamos interessados em garantir que o software previsto é possível. Se houver pouca inovação na tecnologia escolhida ou pouca complexidade com o sistema, esta tarefa não demorará. Se você estiver incluindo um sistema existente, a tarefa poderá não ser necessária, caso não haja necessidade de nenhuma mudança na arquitetura existente. Quando há riscos arquiteturais reais existentes, não deixe a arquitetura se arriscar.

  Como parte dessa atividade, você pode desempenhar alguma seleção de componente e tomar decisões de compra/construção/reutilização. Se isso exigir muito esforço, será possível passar para uma atividade separada.

- **Redefinir a Visão** — Durante a fase de Iniciação, uma Visão foi desenvolvida. Conforme a viabilidade do projeto é determinada e os investidores determinam o tempo de revisão e comentário no sistema, pode haver alterações no documento e em requisitos da Visão. Normalmente, as revisões realizadas nela e os requisitos ocorrem durante a

Elaboração. No final da Elaboração, é estabelecida uma compreensão sólida dos casos de uso mais críticos que conduzem as decisões de arquitetura e planejamento. Os investidores precisam concorda que a visão atual poderá ser atendida se o plano atual for executado para desenvolver o sistema completo, no contexto da arquitetura atual. A quantidade de mudanças deve diminuir durante as iterações subseqüentes, mas haverá a necessidade de alocar algum tempo em cada iteração para o gerenciamento de requisitos.

- **Criar e estabelecer a linha de base de planos de iterações para a fase de Construção** — Preencha os detalhes do plano agora. No final de cada iteração de Construção, revisite os planos e ajuste conforme necessário. Geralmente, os ajustes ocorrem porque o esforço foi incorretamente estimado, o ambiente de negócios foi alterado ou os requisitos foram alterados. Dê prioridade aos esforços de casos de uso, de cenários e técnicos e, em seguida, designe-os às iterações. No final de cada iteração, programe-se para ter um produto de trabalho que forneça o valor para seus investidores.

É possível desempenhar outras atividades durante a Elaboração. Recomenda-se estabelecer o ambiente de teste e iniciar o desenvolvimento dos testes. Enquanto o código detalhado não pode existir, ainda é possível projetar e, talvez, implementar os testes de integração. Os programadores devem estar prontos para desenvolver os testes de unidades e saber como utilizar as ferramentas de teste selecionadas para o projeto. O XP recomenda a gravação do teste antes do código. Essa é uma boa idéia, especialmente, quando está sendo incluída em um corpo do código existente. Entretanto, você escolhe fazer seu teste, o tempo para estabelecer um sistema de teste comum está na Elaboração.

A fase de Elaboração descrita pelo RUP, contém os elementos das fases de Exploração e de Confirmação do XP. A abordagem do XP para tratar de riscos técnicos, como por exemplo a inovação e a complexidade, é a solução “precisa”, isto é, dedicar algum tempo ao experimento para estimar o esforço. Quando há um grande risco não incluído em um caso de uso ou história simples, essa técnica é efetiva em muitos casos, que você precisa aplicar um pouco mais de esforço para garantir o sucesso do sistema e estimativa de esforço precisa.

Geralmente, é possível atualizar artefatos, como por exemplo a Lista de Requisitos e de Riscos da fase de Iniciação, durante a Elaboração. Os artefatos que podem aparecer durante a Elaboração são:

- **SAD (Software Architecture Document)** — O SAD é um artefato composto que fornece uma única origem de informações técnicas no decorrer do projeto. No final da Elaboração, pode conter as descrições detalhadas, do ponto de vista da arquitetura, para casos de uso significativos e identificação de mecanismos chave e elementos de design. Quando o projeto melhora um sistema existente, é possível utilizar um SAD anterior ou decidir se há pouco risco pelo fato de não ter o documento. Em todos os casos, é necessário considerar os processos que lidam com o documento.
- **Planos de Iteração para Construção** — Planeje o número de iterações de Construção durante a Elaboração. Cada iteração possui casos de uso, cenários e outros itens de trabalho específicos designados a ela. Essa informação é capturada e é criada a linha de base nos planos de iteração. Reveja e aprove os planos como parte dos critérios de saída para a Elaboração.

Em projetos muito pequenos e breves, é possível mesclar a iteração de Elaboração com Iniciação e Construção. As atividades essenciais ainda são desempenhadas, mas os recursos para planejamento e revisões de iteração são reduzidos.

Modelo de Caso de Uso Inicial

Enquanto isso pode parecer formal e assustador, é bem direto. Os casos de uso correspondem às “histórias” escritas pelo cliente no XP. A diferença é que um caso de uso é um conjunto completo de ações iniciadas por um autor, alguém ou algo fora do sistema que fornece valor visível. O caso de uso pode conter várias histórias do XP. Para definir o escopo do projeto, o RUP recomenda a identificação dos casos de uso e atores durante a Iniciação. O foco em conjuntos completos de ações do ponto de vista do usuário, ajuda na partição do sistema em partes que fornecem o valor. Isso ajuda a determinar os recursos de implementação apropriados, de modo que, haverá algo para entregar ao cliente no final de cada iteração (possivelmente, exceto as iterações de Iniciação e Elaboração anteriores).

Tanto o RUP como o XP ajudam a assegurar que ainda não chegamos a concluir 80% do sistema inteiro, mas nada foi concluído da forma distribuível. Sempre é necessário conseguir liberar o sistema, para fornecer algum valor ao cliente.

Nesse momento, o modelo de caso de uso identifica os casos de uso e atores com pouco ou nenhum suporte de detalhe. Pode ser texto simples ou diagramas UML (Unified Modeling Language) desenhados à mão ou com uma ferramenta para desenho.

Esse modelo nos ajuda a garantir que foram incluídos os recursos corretos para os investidores, que nada foi esquecido e nos permite visualizar facilmente todo o sistema. É dada prioridade aos casos de uso com base em vários fatores, como por exemplo risco, importância ao cliente e dificuldade técnica.

Nenhum dos artefatos para Iniciação precisa ser excessivamente formal ou grande. Deixe-os tão simples quanto formais, conforme necessário. O XP contém orientação sobre planejamento e aceitação de sistema, enquanto o RUP inclui um pouco mais durante a parte inicial de um projeto. Essa pequena inclusão pode pagar grandes dividendos, considerando um conjunto de riscos mais completo.

Construção

A meta da Construção é concluir o desenvolvimento do sistema. A fase de Construção é, de certa forma, um processo de manufatura, em que a ênfase está no gerenciamento de recursos e controle de operações para otimizar custos, programações e qualidade. Nesse sentido, a intenção do gerenciamento passa por uma transição do desenvolvimento de propriedade intelectual durante a Iniciação e Elaboração, para o desenvolvimento de produtos implantáveis durante a Construção e a Transição.

O XP se concentra na Construção. A fase de Construção é onde o código é produzido. As fases do XP destinam-se a finalidades de planejamento, mas o foco do XP está na construção do código.

Cada iteração de Construção possui três atividades essenciais:

- **Gerenciar recursos e processo de controle** — Todos precisam saber quem vai fazer o que e quando. É necessário certificar-se de que a carga de trabalho não excede a capacidade e que o trabalho está progredindo de acordo com o planejamento.
- **Desenvolver e testar componentes** — Construa os componentes necessários para satisfazer os casos de uso, cenários e outra funcionalidade para a iteração. Eles são testados com os testes de unidades e de integração.
- **Avaliar a iteração** — Na conclusão da iteração, é preciso determinar se as metas da iteração foram atendidas. Se não foram, será preciso dar novamente prioridade e gerenciar o escopo, para atender a data de entrega.

Os tipos diferentes de sistemas requerem técnicas diferentes. O RUP oferece ao engenheiro do software diretrizes diferentes e ajudam a construir os componentes corretos. No formulário de casos de uso e requisitos adicionais (não funcionais), os requisitos são detalhados o suficiente para que o engenheiro realize o trabalho. Várias atividades no RUP fornecem orientação sobre design, implementação e teste de diferentes tipos de componentes. Um engenheiro de software com experiência não precisa analisar essas atividades detalhadamente. Um engenheiro com menos experiência encontrará uma diversidade de ajuda em boas práticas. Cada membro da equipe poderá ou não se destacar no processo, conforme necessário. Entretanto, todos eles visualizam uma única base de conhecimento do processo.

No XP, as histórias levam à implementação. No manual *Extreme Programming Installed*, Jeffries, et al diz que as histórias são "promessas de conversas" com os programadores.[1] A comunicação contínua e efetiva é excelente. Embora sempre há detalhes que precisam de esclarecimento, se as histórias não fossem detalhadas o suficiente para que os programadores realizassem a maior parte de seu trabalho, eles não ficariam prontos. Os casos de uso devem ser detalhados o suficiente, para que os programadores implementem o caso. Em muitos casos, os programadores ajudam a escrevem os detalhes técnicos do caso de uso. Jeffries, et al também diz que as conversas serão documentadas e anexadas na história. O RUP também sugere isso, exceto no formulário de uma especificação de caso de uso, que pode ser tão informal tanto necessário. A captura e o gerenciamento do resultado das conversas é uma tarefa que você deve gerenciar.

O ponto forte do XP é a Construção. Há algumas parte sobre conhecimento e orientação apropriadas para a maioria das equipes. As práticas mais notáveis do XP são:

- **Teste** — Os programadores continuamente escrevem testes para concordar com seu código. Os testes refletem as histórias. O XP o encoraja a escrever os testes primeiro, que é uma prática excelente, porque força fortemente o entendimento das histórias e a fazer mais perguntas onde for necessário. Quer eles sejam escritos antes ou depois da codificação, escreva-os! Inclua-os no conjunto de teste e certifique-se de executá-los toda vez que o código for mudado.

[1] Essa descrição é atribuída a Allistair Cockburn.

- **Reformulação** — Reestrutura o sistema continuamente, sem alterar seu comportamento, para torná-lo mais simples ou incluir flexibilidade. Você precisa determinar se isso é recomendável para sua equipe. O que é simples para uma pessoa, pode ser complexo para outra. Há um exemplo onde dois engenheiros muito competentes em um projeto, passaram a noite toda reescrevendo outro código, porque eles o achavam muito complexo. Como um subproduto, eles interrompiam continuamente as construções no dia seguinte, para o restante da equipe. Testando ajudas, mas a equipe teria sido melhor se não tivessem sido enfrentados conflitos de codificação.
- **Programação em Pares** —O XP reivindica que a programação em pares produz um código melhor em menos tempo. Há evidência de que este é o caso.[1] Há muitos fatores humanos e ambientais a serem considerados, se você implementar essa prática. Os programadores estão querendo tentar isso? Seu ambiente físico comporta dois programadores para trabalhar de maneira eficiente em uma única estação de trabalho? O que você faz com os programadores que estão telecomutando ou que estão em outros locais?
- **Integração Contínua** — Integra e constrói o sistema freqüentemente, talvez, mais de um dia. Essa é uma ótima maneira de assegurar a integridade estrutural do código e permitir o monitoramento de qualidade contínuo através do teste de integração.
- **Direito à Propriedade Coletiva** — Ninguém tem permissão de alterar nenhum código a qualquer hora. O XP conta com o fato de que um bom conjunto de testes de unidades diminuirá o risco dessa prática. Os benefícios de ter todos familiarizados com tudo não scale beyond some point—ten thousand lines of code; twenty thousand; surely less than fifty thousand?
- **Design Simples** — Assim como a recriação, o design do sistema é continuamente modificado para remover a complexidade. Novamente, é necessário determinar a extensão que a escala pode ser aumentada, antes que ela não funcione. Se você perder tempo durante a Elaboração, projetando a arquitetura, acreditamos que o design simples surgirá e se tornará estável o quanto antes.
- **Padrões de Codificação** — São sempre recomendados. Não importa quais são os padrões, contanto que você os tenha e que todos concordem em utilizá-los.

RUP e XP concordam que é necessário gerenciar (ou orientar) iterações. As métricas podem fornecer informações de bom planejamento, visto que o ajuda a escolher o que é melhor para sua equipe. Há três itens para medir: tempo, tamanho e defeitos. A partir daí, é possível obter todas as classificações de estatísticas interessantes. O XP fornece métricas simples a serem utilizadas para determinar o progresso e prognosticar a realização. Essas métricas são centralizadas ao redor do número de histórias concluídas, do número de testes transmitidos e das tendências nesta estatística. O XP cria um caso forte para utilizar uma quantidade mínima de métricas, visto que a visualização de mais não necessariamente aprimorará as oportunidades de sucesso do projeto. O RUP fornece diretrizes sobre quais é possível medir e como medir, e oferece exemplos de métricas. Em todos os casos, as métricas devem ser simples, objetivas, fáceis de coletar, fáceis de interpretar e difíceis de interpretar incorretamente.

Quais artefatos aparecem durante as iterações de Construção? Dependendo se a iteração é uma iteração de Construção anterior ou posterior, é possível criar qualquer um dos itens a seguir:

- **Componente** — Um componente representa uma parte do código do software (origem, binário ou executável) ou um arquivo que contém as informações; por exemplo, um arquivo de inicialização ou um arquivo LEIA-ME. Um componente também pode ser um agregado de outros componentes, como por exemplo um aplicativo que consiste em vários executáveis.
- **Materiais de Treinamento** — Com base nos casos, produza um rascunho preliminar de manuais de usuários e outros materiais de treinamento o mais cedo possível, se o sistema tiver um aspecto forte de interface com o usuário.
- **Plano de Implantação** — O cliente precisa de um sistema. O Plano de Implantação descreve o conjunto de tarefas necessárias para instalar, testar e, efetivamente, transferir o produto para a comunidade de usuários. Para sistemas centrais da Web, foi descoberto que o Plano de Implantação tem maior importância.
- **Plano de Iteração da Fase de Transição** — Conforme o tempo é abordado para implantação do software para os usuários, conclua e revise o Plano de Iteração da Fase de Transição.

---

[1] *Strengthening the Case for Pair Programming*, IEEE Software, July/August, 2000.

Na verdade, tudo isso é sobre o código?

Além das diferenças na abordagem da arquitetura entre o RUP e o XP, há outras. Uma é a forma com que o design é comunicado. O XP indica que o código é o design e o design é o código. De fato, o código está sempre em conformidade com ele mesmo. Acredita-se que algum esforço para capturar e comunicar o design, que não seja o código, é um tempo bem aproveitado. Essa pequena história pode esclarecer isso.

Um engenheiro tinha duas experiências em projetos de software, onde o design estava no código e era o único lugar para localizar informações de design. Ambos esses projetos eram relatados pelo compilador: um era para melhorar e manter um otimizador para um compilador Ada e o outro era um projeto para transferir o front-end de um compilador para uma nova plataforma e vincular um gerador de código de terceiros a ele.

A compilador do compilador é complicada, mas bem conhecida. Em ambos os projetos, o engenheiro queria uma visão geral do design e a implementação do compilador (ou otimizador). Em cada caso, ele recebeu uma quantidade de listagens de códigos-fonte, escala de várias polegadas e foi dito para "procurar lá." A ele teria sido oferecida qualquer coisa por alguns programas bem construídos com algum texto de suporte. O projeto do otimizador nunca foi concluído. O projeto do compilador foi concluído com êxito, com qualidade de código excelente, por causa de um amplo conjunto de testes desenvolvido juntamente com o código. O engenheiro passou dias acompanhando o código em um depurador, para tentar entender o que ele fazia. O custo pessoal de destruições menores e o custo para a equipe não valia a pena. Não tínhamos a opção de parar depois de 40 horas como o XP sugeria e fizemos um esforço heróico para que a tarefa fosse concluída.

O problema principal em ter apenas o código é que o código—independentemente de estar bem documentado—não indica o problema que ele resolve, ele apenas comunica a solução para o problema. Alguma documentação dos requisitos percorre um longo caminho para explicar os objetivos originais depois de ter sido movida pelos usuários originais e pelos desenvolvedores. Para manter um sistema, geralmente você precisa saber o que a equipe do projeto original tem em mente. Alguns documentos de design de alto nível são semelhantes – geralmente o código está em um nível de abstração muito baixo para indicar realmente o que o sistema, como um todo, está tentando fazer. Isso é especialmente verdade em sistemas Orientados por objetos no qual o encadeamento de execução é difícil ou impossível de seguir, simplesmente pela análise das classes. Os documentos de design o guia sobre onde olhar quando há problemas posteriormente —e isso sempre*acontece*.

A moral da história é que gastar algum tempo capturando e mantendo documentos de design realmente ajuda. Reduz o risco de incompreensão e pode agilizar o desenvolvimento. A abordagem XP é gastar alguns minutos esboçando o design ou para usar cartões CRC.[1] A equipe não mantém isso, mas trabalhará no código. Há uma hipótese implícita de que as tarefas são simples o suficiente, que já sabemos como continuar. Mesmo se soubermos, a próxima pessoa a aparecer pode não ser tão sortuda. O RUP sugere que um pouco mais de tempo seja dispensado na captura e manutenção desses artefatos de design.

Transição

O foco da Transição é assegurar que o software esteja disponível para seus usuários finais. A fase de Transição inclui testar o produto em preparação para release e ajustes pequenos com base no feedback do usuário. Nesse momento no ciclo de vida, o feedback do usuário precisa principalmente ser focalizado no ajuste do produto, na configuração, na instalação e nos problemas de utilidade.

O release inicial e freqüente é uma ótima idéia. Mas, o que você quer dizer com versão? O XP é incerto sobre isso e não trata dos problemas de fabricação necessários para liberar o software comercial. Você pode conseguir diminuir alguns dos problemas em um projeto interno; mas até então, precisa da documentação, treinamento e assim por diante. E sobre o gerenciamento de suporte e de mudanças? É real esperar que o cliente on-site controlará isso também? Bruce Conrad enfatizou em sua revisão do InfoWorld do XP[2] que os clientes não querem adquirir o software que está continuamente em mudança. É necessário ponderar o benefício de conseguir mudanças para o cliente rapidamente, com as desvantagens de mudanças e possível desestabilização.

Quando você decide liberar, é preciso fornecer ao usuário final mais do que apenas o código. As atividades e os artefatos no guia de Transição, o conduzem a esta parte do processo de desenvolvimento de software. As atividades são centradas na obtenção de um produto utilizável para seu cliente. As atividades de Transição essenciais são as seguintes:

---

[1] Os cartões CRC (Classe, Responsabilidade e Colaboração), desenvolvidos por Kent Beck e Ward Cunningham, ensinam aos profissionais os princípios de design Orientado por Objetos..

[2] http://www.infoworld.com/articles/mt/xml/00/07/24/000724mtextreme.xml

Field Code Changed

- **Finalizar Material de Suporte ao Usuário Final** — Essa atividade pode ser tão simples quanto verificar os itens fora de uma lista, mas é preciso certificar-se de que sua organização esteja preparada para suportar seu cliente.
- **Testar o Produto Distribuível em um Ambiente do Cliente** —Se você puder simular o ambiente do cliente em seu local, faça isso. Do contrário, vá até o cliente e instale o software para garantir seu funcionamento. Você não quer estar na difícil posição de dizer ao cliente "mas funcionou em nosso sistema".
- **Ajustar o Produto Com Precisão Baseado no Feedback do Cliente** — Se possível, planeje um ou mais períodos de testes beta, no qual o software é entregue para um número limitado de clientes. Se fizer isso, será preciso gerenciar o período de teste beta e considerar o feedback do cliente em seu "jogo final".
- **Entregar o Produto Final para o Usuário Final** — Dependendo do tipo de produto e release do software, há muitos detalhes sobre pacote, fabricação e outros problemas de produção para se tratar. Raramente você apenas coloca o software em um diretório e envia um correio, permitindo que sua comunidade de clientes saiba que o software está disponível para eles.

Como na maioria das outras fases, a quantia de formalidade e complexidade de seu processo variará. Contudo, se você não prestar atenção aos detalhes da implantação, poderá impedir semanas e meses de esforço de bom desenvolvimento e terminar com um produto deficiente no marketplace de destino.

Você pode produzir diversos artefatos durante a fase de Transição. Se o seu produto for um que terá versões futuras (e quantos não tem?), você terá que começar a identificar os recursos e a detectar correções para a próxima versão. Os artefatos essenciais para qualquer projeto são:

- **Plano de Implantação** — Conclua o Plano de Implantação iniciado na fase de Construção e utilize-o como roteiro para entrega ao cliente.
- **Notas Sobre o Release** — É um produto de software raro que não tem as instruções mais recentes para o usuário final. Planeje-o e tenha um formato útil e consistente para suas notas.
- **Materiais de Treinamento e Documentação** — Existe um amplo espectro de formulários que podem ser abordados por esses materiais. Você fornecerá tudo on-line? Terá um tutorial? A ajuda do produto é completa e útil? Não ache que o cliente saberá que você sabe. Seu sucesso depende da ajuda satisfatória fornecida.

Sumário

A construção de um software é mais do que um código escrito. Um processo de desenvolvimento de software deverá dar enfoque a todas as atividades necessárias para oferecer qualidade aos seus clientes. Um processo completo não tem que ser pesado. Mostramos como você pode ter um processo pequeno e completo, dando enfoque a atividades e artefatos essenciais para o projeto. Desempenhe uma atividade ou produza um artefato se isso ajudar a mitigar o risco em seu projeto. Utilize a quantidade de processo e formalidade necessária para a equipe de projeto e a organização.

O RUP e XP não são necessariamente exclusivos. Incorporando técnicas de ambos os métodos, você poderá chegar a um processo que o ajudará a fornecer um software de melhor qualidade mais rápido do que o que você tem hoje. Robert Martin descreve um processo chamado *processo dX*, que ele afirma ser compatível com RUP.[1] É uma instância de um processo criado a partir da estrutura RUP.

Um bom processo de software incorpora as boas práticas comprovadas do mercado. As boas práticas são aquelas que foram testadas com o passar do tempo, em organizações reais de desenvolvimento de software. O XP é um método que é foco de muita atenção hoje. É centrado em código e oferece uma promessa de produtividade máxima e código extra de processo mínima. Há muitas técnicas no XP que garantem a consideração e a adoção nas situações certas.

O XP enfoca histórias, testes e código—aborda o planejamento até uma determinada extensão, mas trata a captura de planos levemente. O XP implica que você pode produzir outras coisas como "fazer um design CRC com alguns cartões ou projetar um UML…" ou "Não produzir documentos ou outros artefatos que não estão sendo utilizados …", mas são tratados na transmissão. O RUP o convida a considerar a produção de apenas o que é útil e necessário a medida que você formula e atualiza o plano de desenvolvimento e identifica o que essas coisas poderiam ser.

[1] http://www.objectmentor.com/publications/RUPvsXP.pdf. Este é um capítulo de um livro não publicado por Martin, Booch e Newkirk.

Field Code Changed

RUP é um processo que indica o ciclo de vida do desenvolvimento do software completo. Ele dá enfoque às boas práticas envolvidas em milhares de projetos. Nós incentivamos a investigação e a invenção de novas técnicas que levam às boas práticas. À medida que surgem as novas boas práticas, procuramos incorporá-las no RUP.

Apêndice A: Rational Unified Process

O Rational Unified Process ou RUP fornece uma abordagem disciplinada de desenvolvimento de software. É um *produto de processo*, desenvolvido e mantido pelo Rational Software. Ele vem com vários roteiros disponíveis para tipos diferentes de projetos de software. O RUP também fornece informações para ajudá-lo a usar outras ferramentas Rational para desenvolvimento de software, mas essas ferramentas não são obrigatórias para um aplicativo eficiente para uma organização; integrações com ofertas de outros fornecedores também são possíveis.

O RUP oferece orientação para todos os aspectos de um projeto de software. Ele não exige que você desempenhe quaisquer atividades específicas nem produza qualquer artefato específico. Ele fornece informações e orientações para que você decida sobre o que é aplicável para a sua organização. Também fornece orientações que o ajudam a adequar o processo se nenhum dos roteiros disponíveis se ajustar ao seu projeto ou organização.

O RUP enfatiza a adoção de determinadas *boas práticas* do desenvolvimento de software moderno, como uma forma de reduzir o *risco* inerente ao desenvolvimento de novo software. Essas boas práticas são:

1. Desenvolver iterativamente
2. Gerenciar requisitos
3. Utilizar arquiteturas com base no componente
4. Modelar visualmente
5. Verificar continuamente a qualidade
6. Controlar alterações

Formatted: Bullets and Numbering

Essas boas práticas são elaboradas nas definições do Rational Unified Process de:

- *Funções* — conjuntos de tarefas executadas e artefatos elaborados.
- *Disciplinas* — áreas de enfoque de esforço de engenharia de software como Requisitos, Análise e Design, Implementação e Teste.
- *Tarefas* — definições da forma que os artefatos são produzidos e avaliados.
- *Artefatos* — os produtos de trabalho utilizados, produzidos ou modificados no desempenho das tarefas

O RUP é um processo iterativo que identifica quatro fases de qualquer projeto de desenvolvimento de software. Com o passar do tempo, o projeto passa pelas fases de Iniciação, Elaboração, Construção e Transição. Cada fase contém uma ou mais iterações na qual você produz um executável, mas talvez um sistema incompleto (exceto possivelmente na fase de Iniciação). Durante cada iteração, você desempenha as atividades de várias disciplinas em níveis variáveis de detalhes. A seguir, um diagrama de visão geral do RUP.

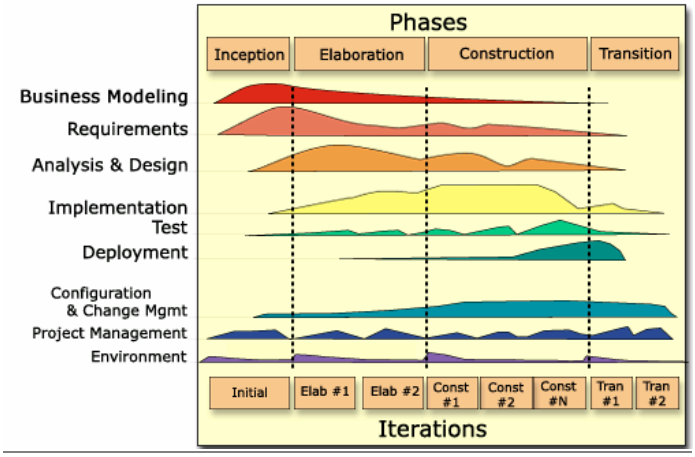


Diagrama de Visão Geral do RUP

O Rational Unified Process, An Introduction, Segunda Edição é uma boa visão geral do RUP. Você pode localizar mais informações e uma avaliação do RUP no site do Rational Software em: www.rational.com.

Field Code Changed

Apêndice B: eXtreme Programming

eXtreme Programming (XP) é uma disciplina de desenvolvimento de software desenvolvida por Kent Beck em 1996. Ele baseia-se em quatro valores: comunicação, simplicidade, feedback e coragem. Enfatiza a comunicação contínua entre o cliente e os membros da equipe de desenvolvimento, tendo um cliente on-site enquanto o desenvolvimento progride. O cliente on-site decide o que será construído e em que ordem. Você incorpora a simplicidade, recriando continuamente o código e produzindo um conjunto mínimo de artefatos sem código. Muitas versões pequenas e testes de unidades contínuos são mecanismos de feedback. Coragem significa fazer a coisa certa, mesmo quando não é a coisa mais comum a ser feita. Significa ser honesto sobre o que você pode ou não.

Doze práticas XP suportam os quatro valores. São elas:

- **Jogo de planejamento** — Determina os recursos na próxima versão por uma combinação de histórias priorizadas e estimativas técnicas.
- **Versões pequenas** — Libera o software com freqüência para o cliente com pequenas versões incrementais.
- **Metáfora** — A metáfora é uma história ou descrição simples compartilhada de como o sistema funciona.
- **Design simples** — Mantenha o design simples, mantendo o código simples. Procure continuamente pela complexidade no código e remova-a imediatamente.
- **Teste** — O cliente grava os testes para testar as histórias. Os programadores gravam os testes para testar tudo que possa causar falha no código. Você grava testes antes de gravar o código.
- **Recriação** — Esta é uma técnica de simplificação para remover a duplicação e a complexidade do código.
- **Programação em pares** — Equipes de dois programadores em um único computador desenvolver todo o código. Um grava o código ou *conduz*, enquanto o outro revisa o código para que haja correção e capacidade de entendimento ao mesmo tempo.
- **Propriedade coletiva** — O código é de todos. Isso significa que todos tem a capacidade de alterar qualquer código a qualquer momento.

- **Integração contínua** — Cria e integra o sistema várias vezes por dia sempre que alguma tarefa de implementação é concluída.
- **Semana de 40 horas** — Os programadores não podem trabalhar de maneira eficiente se estiverem cansados. As horas extras nunca são permitidas por duas semanas consecutivas.
- **Cliente on-site** — Um cliente real trabalha no ambiente de desenvolvimento o tempo todo para ajudar a definir o sistema, a escrever testes e a responder perguntas.
- **Padrões de codificação** — Os programadores adotam um padrão de codificação consistente.

No momento, há três manuais disponíveis que descrevem o XP:

1. eXtreme Programming Explained
2. Extreme Programming Installed
3. Planning Extreme Programming

Formatted: Bullets and Numbering

Vários sites estão disponíveis para obter mais informações sobre o XP.

Rational®
the software development company

Duas Sedes:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Sem custo: (800) 728-1212
E-mail: info@rational.com
Web: www.rational.com
Localização Internacional: www.rational.com/worldwide

Field Code Changed

Field Code Changed

Field Code Changed