O **ciclo de vida de software** é uma série de etapas que um software passa, desde a sua concepção inicial até a sua desativação. Ele funciona como um guia, ajudando as equipes a planejar, projetar, desenvolver, testar, implantar e manter o produto de forma organizada.

Existem diferentes modelos de ciclo de vida, mas o modelo **Cascata** (Waterfall) e o modelo **Ágil** (Agile) são dois dos mais conhecidos, com diferenças fundamentais.

Modelo Cascata (Waterfall)

O modelo Cascata é uma abordagem **linear e sequencial**. As fases do projeto (como coleta de requisitos, análise, design, implementação e testes) são executadas uma após a outra, como uma cachoeira. Uma fase deve ser concluída e validada antes que a próxima comece.

- **Características Principais:**
  - **Sequencial:** O fluxo de trabalho é rígido e segue uma ordem predefinida.
  - **Documentação Pesada:** Exige um planejamento detalhado e uma documentação completa no início do projeto.
  - **Pouca Flexibilidade:** É difícil e caro fazer mudanças nos requisitos após o início do desenvolvimento.

Modelo Ágil (Agile)

O modelo Ágil é uma abordagem **iterativa e incremental**. O projeto é dividido em ciclos curtos e repetitivos (chamados de "sprints"), onde pequenas partes do software são desenvolvidas e entregues. A equipe trabalha em estreita colaboração com o cliente para obter feedback constante e se adaptar a mudanças.

- **Características Principais:**
  - **Iterativo:** O software é desenvolvido em incrementos, com novas funcionalidades sendo adicionadas a cada ciclo.
  - **Flexibilidade:** Permite e até incentiva mudanças nos requisitos, mesmo em fases avançadas.
  - **Colaboração:** A comunicação e o feedback do cliente são contínuos durante todo o projeto.

Principais Diferenças

Característica	**Cascata (Waterfall)**	**Ágil (Agile)**
Abordagem	Linear e Sequencial	Iterativa e Incremental
Mudanças	Difíceis e Custosas	Flexíveis e Bem-vindas
Entrega	Única, no final do projeto	Frequentes e em pequenos incrementos
Participação do Cliente	Concentrada no início do projeto	Constante durante todo o projeto
Documentação	Essencial e detalhada no início	Enfatiza menos a documentação e mais o software funcional
Tolerância a Erros	Erros só são descobertos no final do projeto	Erros são identificados e corrigidos em cada ciclo

O diagrama do ciclo de vida do software varia de acordo com a metodologia utilizada. Os dois modelos mais comuns, o **Cascata** e o **Ágil**, podem ser visualizados de maneiras distintas para mostrar como o trabalho flui em cada um.

Modelo Cascata (Waterfall)

Este diagrama mostra o fluxo **linear** e **sequencial** do modelo Cascata. Cada fase é concluída e validada antes que a próxima comece, sem retorno.

Modelo Ágil (Agile)

Este diagrama representa a abordagem **iterativa e cíclica** do modelo Ágil. O processo se repete em ciclos curtos (sprints) que entregam funcionalidades incrementais e recebem feedback contínuo.

Ciclo de desenvolvimento Ágil de software

Lucid Content

Tempo de leitura: cerca de 4 minutos

As 6 fases do ciclo de vida de desenvolvimento Ágil

Elabore um escopo e priorize projetos

Crie diagramas para os requisitos do sprint inicial

Desenvolvimento/iteração

Comece a produzir a iteração

Produção e suporte contínuo para a versão do software

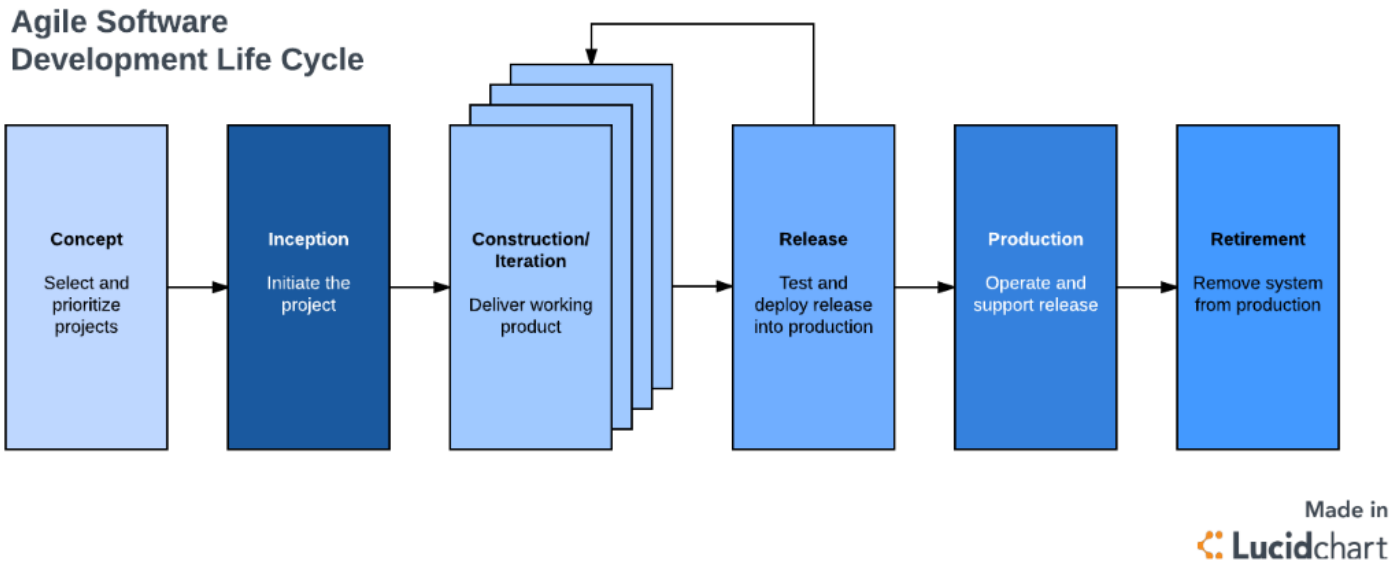
Descontinuar

Inovar ou morrer.

Se as empresas de tecnologia quiserem manter sua relevância em um mercado acelerado e dinâmico, suas equipes de desenvolvimento de software precisam saber como impulsionar seus produtos o máximo possível, e de maneira rápida. A metodologia Ágil de desenvolvimento de software foi desenvolvida especificamente para facilitar o desenvolvimento e implementação rápidos de software.

Conheça as fases do ciclo de vida de desenvolvimento Ágil de software (SDLC, em inglês) para saber se esse processo atenderá às necessidades da sua equipe.

ciclo de vida de desenvolvimento de software ágil



Visão geral do ciclo de vida de desenvolvimento de software Ágil (clique na imagem para modificar on-line)

1. Elabore um escopo e priorize projetos

Durante a primeira fase do ciclo de vida de desenvolvimento Ágil de software, a equipe elabora um escopo e prioriza projetos. Algumas equipes podem trabalhar em mais de um projeto ao mesmo tempo, dependendo da organização da área.

Para cada conceito, defina a oportunidade comercial e determine o tempo e esforço necessários para concluir o projeto. Com base nessas informações, você pode avaliar a viabilidade técnica e econômica e decidir quais projetos vale a pena desenvolver.

2. Crie diagramas para os requisitos do sprint inicial

Depois de decidir o projeto, trabalhe com as partes interessadas para determinar os requisitos. Recomendamos usar diagramas de fluxo de usuário ou diagramas UML de alto nível para demonstrar como o novo recurso funcionará e como ele fará parte do seu sistema existente.

fluxo da trajetória do usuário

Modelo de fluxo de jornada do usuário (clique na imagem para modificar on-line)

Em seguida, escolha os membros da equipe que trabalharão no projeto, e maneje seus recursos. Crie um cronograma ou um mapa de processo de raias no Lucidchart para delegar responsabilidades e mostrar com clareza os prazos de conclusão de trabalhos durante o sprint.

Por exemplo, nossa equipe de produtos criou o diagrama abaixo para visualizar como a equipe implementaria o processo de impressão e envio de uma empresa. As colunas mostram a carga de trabalho de cada membro da equipe, e as linhas mostram o trabalho concluído em cada sprint.

plano de lançamento

3. Desenvolvimento/iteração

Depois que a equipe definir os requisitos do sprint inicial com base nas opiniões e nos requisitos das partes interessadas, o trabalho realmente começará. Designers e desenvolvedores de UX começam a desenvolver sua primeira iteração do projeto, com a meta de ter um produto funcional para lançar no final do sprint. Lembre-se de que o produto passará por diversas revisões, portanto, essa primeira iteração pode incluir apenas um mínimo de funcionalidades. A equipe terá outros sprints para desenvolver mais o produto.

4. Comece a produzir a iteração

Você está quase pronto para lançar seu produto ao mundo. Siga os seguintes passos para concluir a iteração do software:

Teste o sistema. Sua equipe de garantia de qualidade (QA) precisa testar as funcionalidades, detectar bugs e registrar pontos positivos e negativos.

Solucione os defeitos.

Finalize a documentação do sistema e do usuário. Use diagramas UML do Lucidchart para visualizar seu código ou para demonstrar fluxos de usuário, ajudando todos a entenderem como o sistema funciona e como podem desenvolvê-lo mais.

Comece a produzir a iteração.

5. Produção e suporte contínuo para a versão do software

Essa fase é para fornecer suporte contínuo para a versão do software. Ou seja, sua equipe precisa manter um funcionamento contínuo do sistema e mostrar aos usuários como usá-lo. A fase de produção termina quando o suporte é encerrado, ou quando há uma data de descontinuação da versão.

6. Descontinuar

Durante a fase de descontinuação, encerre a produção da versão do sistema — normalmente quando você quer substituir um sistema por uma nova versão ou quando o sistema se torna redundante, obsoleto ou se opõe ao seu modelo de negócios.

Planejamento da sprint de desenvolvimento Ágil de software

Dentro do SDLC Ágil, o trabalho é dividido em sprints, com o objetivo de criar um produto funcional no final de cada sprint. Um sprint normalmente dura duas semanas, ou 10 dias úteis. O fluxo de trabalho de um sprint deve seguir este resumo básico:

Planejamento. O sprint começa com uma reunião de planejamento sprint, em que membros da equipe se reúnem para definir os componentes da rodada de trabalho. O gerente de produto prioriza trabalhar o backlog de tarefas, e as atribui à equipe.

Desenvolvimento. Projete e desenvolva o produto de acordo com as orientações aprovadas.

Teste/controle de qualidade. Faça testes rigorosos e documente os resultados antes da entrega.

Entrega. Apresente o produto ou software em pleno funcionamento às partes interessadas e aos clientes.

Avaliação. Solicite opiniões e informações do cliente e das partes interessadas para incorporar no próximo sprint.

Além das reuniões de planejamento sprint, recomendamos fazer encontros diários com sua equipe para acompanhar o andamento, solucionar conflitos e dar continuidade ao processo.

Seja flexível e aberto a mudanças. Afinal, a metodologia chama-se “Ágil” por um bom motivo.

Conclusão: o objetivo do ciclo de vida de desenvolvimento Ágil de software é criar e entregar software funcional o mais rápido possível.

Ciclos de Vida do Software

Neste contexto, neste artigo apresentaremos alguns modelos de ciclo de vida, quais sejam: Cascata, Modelo em V, Incremental, Evolutivo, RAD, Prototipagem, Espiral, Modelo de Ciclo de Vida Associado ao RUP.

O ciclo de vida é a estrutura contendo processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software, abrangendo a vida do sistema, desde a definição de seus requisitos até o término de seu uso.

O modelo de ciclo de vida é a primeira escolha a ser feita no processo de software. A partir desta escolha definir-se-á desde a maneira mais adequada de obter as necessidades do cliente, até quando e como o cliente receberá sua primeira versão operacional do sistema.

Processo de software é o conjunto de atividades que constituem o desenvolvimento de um sistema computacional. Estas atividades são agrupadas em fases, como: definição de requisitos, análise, projeto, desenvolvimento, teste e implantação.

Em cada fase são definidas, além das suas atividades, as funções e responsabilidades de cada membro da equipe, e como produto resultante, os artefatos.

O que diferencia um processo de software do outro é a ordem em que as fases vão ocorrer, o tempo e a ênfase dados a cada fase, as atividades presentes, e os produtos entregues.

Com o crescimento do mercado de software, houve uma tendência a repetirem-se os passos e as práticas que deram certo. A etapa seguinte foi a formalização em modelos de ciclo de vida.

Em outras palavras, os modelos de ciclo de vida são o esqueleto, ou as estruturas pré-definidas nas quais encaixamos as fases do processo. De acordo com a NBR ISO/IEC 12207:1998, o ciclo de vida é a “Estrutura contendo processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software, abrangendo a vida do sistema, desde a definição de seus requisitos até o término de seu uso.”

O modelo de ciclo de vida é a primeira escolha a ser feita no processo de software. A partir desta escolha definir-se-á desde a maneira mais adequada de obter as necessidades do cliente, até quando e como o cliente receberá sua primeira versão operacional do sistema.

Não existe um modelo ideal. O perfil e complexidade do negócio do cliente, o tempo disponível, o custo, a equipe, o ambiente operacional são fatores que influenciarão diretamente na escolha do ciclo de vida de software a ser adotado.

Da mesma forma, também é difícil uma empresa adotar um único ciclo de vida. Na maior parte dos casos, vê-se a presença de mais de um ciclo de vida no processo.

Os ciclos de vida se comportam de maneira sequencial (fases seguem determinada ordem) e/ou incremental (divisão de escopo) e/ou iterativa (retroalimentação de fases) e/ou evolutiva (software é aprimorado).

Neste contexto, neste artigo apresentaremos alguns modelos de ciclo de vida, quais sejam:

- Cascata
- Modelo em V
- Incremental
- Evolutivo

- RAD
- Prototipagem
- Espiral
- Modelo de Ciclo de Vida Associado ao RUP

Modelo em Cascata

Formalizado por Royce em 1970, é o modelo mais antigo. Suas atividades fundamentais são:

- análise e definição de requisitos;
- projeto;
- implementação;
- teste;
- integração.

O modelo em cascata tem o grande mérito de ser o primeiro a impor o planejamento e o **gerenciamento ao processo de software**, que antes era casual. O nome "cascata" foi atribuído em razão da sequência das fases, onde cada fase só começa quando a anterior termina; e da transmissão do resultado da fase anterior como entrada para a fase atual (o fim de cada fase resulta em um documento aprovado). Nesse modelo, portanto, é dada muita ênfase às fases de análise e projeto antes de partir para a programação, a fim de que o objetivo do software esteja bem definido e que sejam evitados retrabalhos, conforme podemos observar na **Figura 1**.

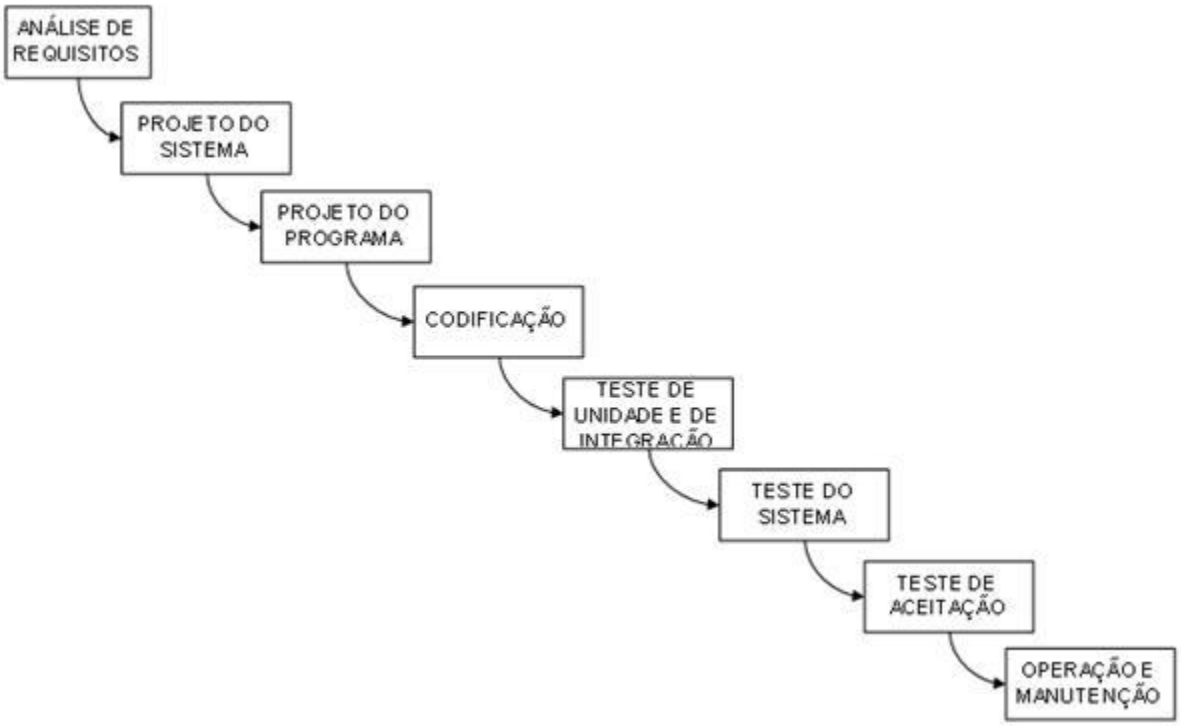


Figura 1. O modelo em cascata

Devido à sua simplicidade, o modelo em cascata é fácil de ser entendido pelo cliente. É um modelo que supõe um início e fim claro e determinado, assim como uma estimativa precisa de custo logo no início, fatores importantes na conquista do cliente.

O problema se dá depois, quando o cliente, após esperar até o fim do processo para receber a primeira versão do sistema, pode não concordar com ela. Apesar de cada fase terminar com uma documentação aprovada, certamente haverá lacunas devido a requisitos mal descritos pelo cliente, mal entendido pelo

analista ou por mudança de cenário na organização que exija adaptação de requisitos. O modelo em cascata não prevê revisão de fases.

Assim, o risco é muito alto, principalmente para sistemas complexos, de grande porte, afinal o modelo em cascata pressupõe uma realidade estática e bem conhecida, comparado a uma linha de produção fabril. Mas a rotina do negócio do cliente não reflete isso. Manipulação de usuários com diferentes habilidades, ambientes operacionais distintos, tecnologia em crescente evolução, necessidade de integração com outros sistemas (em plataformas antigas ou mais novas), mudanças organizacionais, até mudanças na legislação do município/estado/país, pedem um modelo mais flexível.

Por outro lado, o modelo em cascata adéqua-se bem como um "submodelo" para outros modelos. Por exemplo, no modelo "cascata com realimentação" permite-se que, a cada descoberta da fase posterior, haja uma correção da fase anterior.

Modelo em V

Neste modelo, do Ministério de Defesa da Alemanha, 1992, o modelo em cascata é colocado em forma de "V". Do lado esquerdo do V ficam da análise de requisitos até o projeto, a codificação fica no vértice e os testes, desenvolvimento, implantação e manutenção, à direita, conforme **Figura 2**.

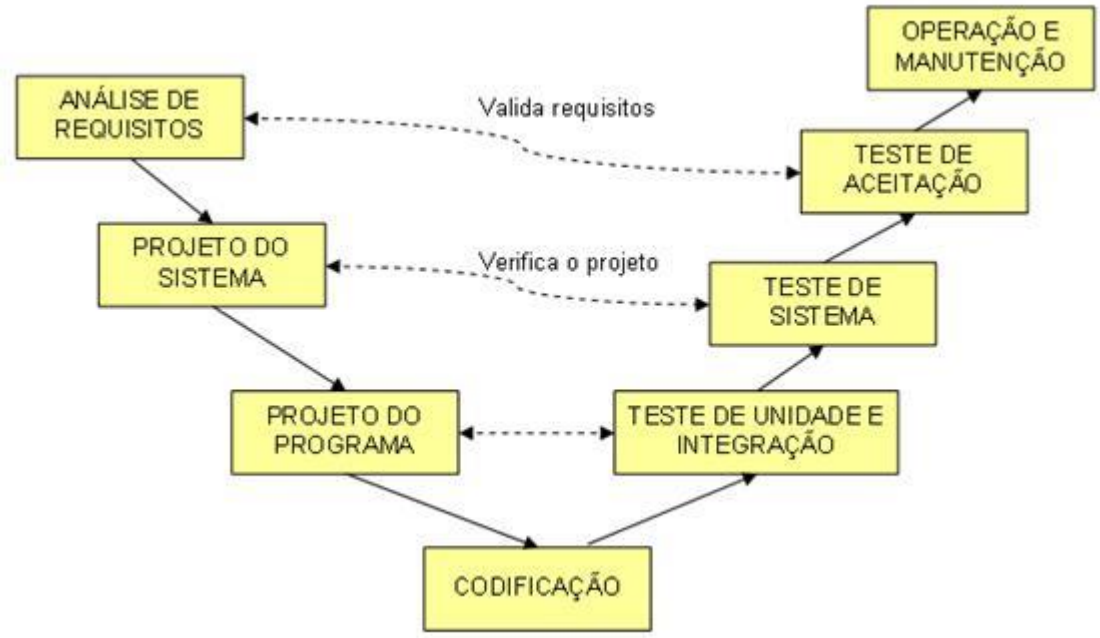


Figura 2. O modelo em V

A característica principal desse modelo, que o diferencia do modelo em cascata, é a ênfase dada à verificação e validação: cada fase do lado esquerdo gera um plano de teste a ser executado no lado direito.

Mais tarde, o código fonte será testado, do mais baixo nível ao nível sistêmico para confirmar os resultados, seguindo os respectivos planos de teste: o teste de unidade valida o projeto do programa, o **teste de sistema** valida o projeto de sistema e o teste de aceitação do cliente valida a análise de requisitos.

Da mesma forma que o modelo em cascata, o cliente só recebe a primeira **versão do software** no final do ciclo, mas apresenta menos risco, devido ao planejamento prévio dos testes nas fases de análise e projeto.

Ciclos de Vida Incremental

Neste modelo, de Mills em 1980, os requisitos do cliente são obtidos, e, de acordo com a funcionalidade, são agrupados em módulos. Após este agrupamento, a equipe, junto ao cliente, define a prioridade em que cada módulo será desenvolvido, escolha baseada na importância daquela funcionalidade ao negócio do cliente.

Cada módulo passará por todas as fases "cascata" de projeto, conforme se observa na **Figura 3**, e será entregue ao cliente um software operacional. Assim, o cliente receberá parte do produto final em menos tempo.

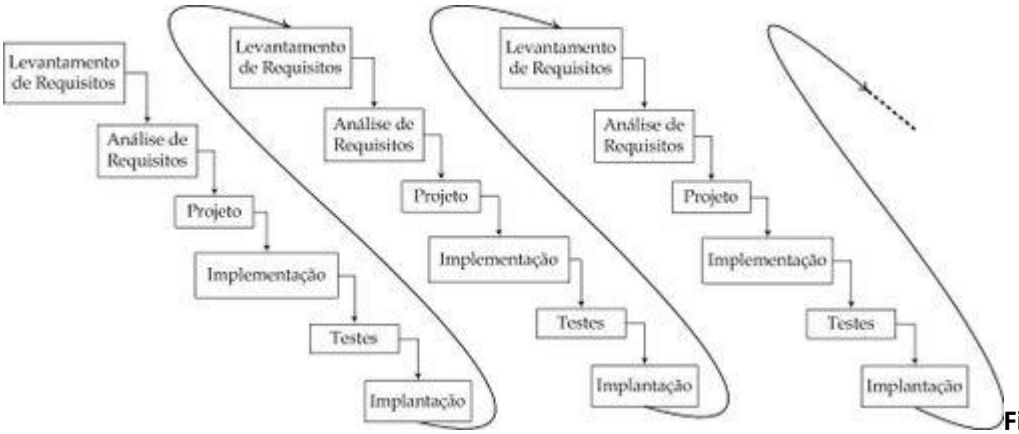


Figura 3. Ciclo de vida Incremental

Como o cliente já trabalhará no primeiro incremento ou módulo, é muito importante que haja uma especial atenção na integração dos incrementos, o que exige muito planejamento, afinal não é aceitável que o cliente se depare com muitos erros de software a cada incremento, tampouco, que a cada incremento ele precise se readaptar a grandes mudanças. Uma atenção especial deve ser dada ao agrupamento dos requisitos e à qualidade no desenvolvimento das funções comuns a todo o sistema, que inevitavelmente deverão ser entregues no primeiro incremento.

Desta forma, além de atender as necessidades mais críticas do cliente mais cedo, as partes mais importantes serão, também, as partes mais testadas no ambiente real. Será mais difícil gastar recursos em conceitos errados, ou que um mau entendimento dos requisitos alcance uma escala difícil de ser ajustada, visto que durante todo o projeto haverá o feedback do cliente (a opinião do cliente realimenta o sistema).

Esse ciclo de vida não exige uma equipe muito grande, pois a modularização diminui o escopo de cada incremento, e não há um paralelismo nas atividades. Haverá, por outro lado, uma dificuldade em manter a documentação de cada fase atualizada devido às melhorias no sistema e aos ajustes de requisitos solicitados pelos clientes.

Modelo Evolutivo

Neste modelo, os requisitos são adquiridos em paralelo à evolução do sistema. O modelo evolutivo parte do princípio que o cliente não expõe todos os requisitos, ou os requisitos não são tão bem conhecidos, ou os requisitos ainda estão sofrendo mudanças. Desta forma, a análise é feita em cima dos requisitos conseguidos até então, e a primeira versão é entregue ao cliente. O cliente usa o software no seu ambiente operacional, e como feedback, esclarece o que não foi bem entendido e dá mais informações sobre o que precisa e sobre o que deseja (ou seja, mais requisitos).

A partir deste feedback, nova análise, projeto e desenvolvimento são realizados, e uma segunda versão do software é entregue ao cliente que, novamente, retorna com mais feedbacks. Assim, o software vai evoluindo, se tornando mais completo, até atender todas as necessidades do cliente dentro do escopo estabelecido. Tem-se assim a versão final, pelo menos até novos requisitos aparecerem (ver **Figura 4**).

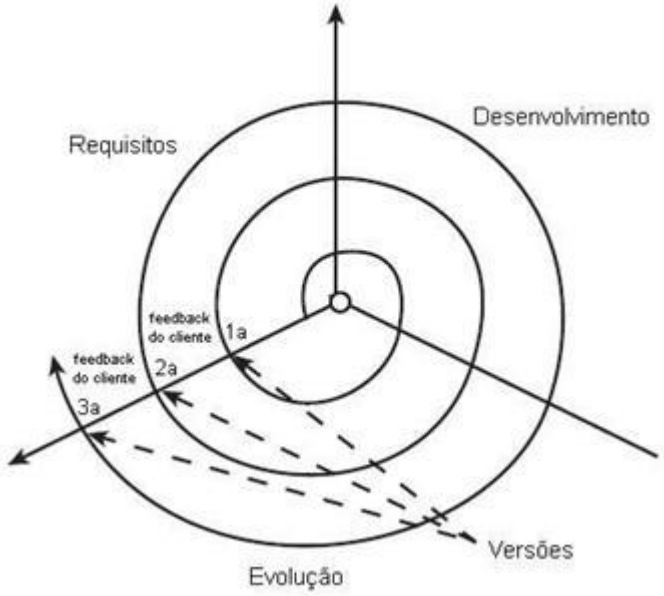


Figura 4. Ciclo de vida Evolutivo

A participação constante do cliente é uma grande vantagem desse modelo, o que diminui o risco de má interpretação de requisitos dos modelos que só oferecem a primeira versão do software no final do processo. Da mesma forma, o software já atende algumas necessidades do cliente muito mais cedo no processo.

Não é dada muita ênfase à documentação, pois a geração de versões torna este trabalho muito árduo. Além disso, como a análise de requisitos e desenvolvimento estão sempre acontecendo, a preocupação em documentar todo o processo pode fazer com que haja atrasos na entrega.

Há uma alta necessidade de gerenciamento nesse tipo de modelo, pois a falta de documentação adequada, o escopo de requisitos não determinado, o software crescendo e estando ao mesmo tempo em produção podem ter consequências negativas. Seguem alguns exemplos: o sistema nunca terminar, pois o cliente sempre pede uma alteração; o sistema não ter uma estrutura robusta a falhas nem propícia a uma fácil manutenção, pelas constantes alterações; o cliente mudar de ideia radicalmente entre uma versão e outra ou revelar um requisito que exija uma versão bem diferente da anterior, fazendo com que toda a base (de dados ou de programação) precise ser revista. Os citados problemas podem implicar em um grande ônus financeiro e de tempo.

É muito importante que o cliente esteja ciente do que se trata este ciclo de vida e que sejam esclarecidos os limites de escopo e de tempo, para que não haja frustrações de expectativas.

RAD – “Rapid Application Development”

Este modelo formalizado por James Martin em 1991, como uma evolução da “prototipagem rápida”, destaca-se pelo desenvolvimento rápido da aplicação. O ciclo de vida é extremamente comprimido, de forma a encontrarem-se exemplos, na literatura, de duração de 60 e 90 dias. É ideal para clientes buscando lançar soluções pioneiras no mercado.

É um ciclo de vida incremental, iterativo, onde é preferível que os requisitos tenham escopo restrito. A diferença principal do ciclo anterior é o forte paralelismo das atividades, requerendo, assim, módulos bastante independentes. Aqui os incrementos são desenvolvidos ao mesmo tempo, por equipes diferentes.

Além do paralelismo, a conquista do baixo tempo se dá graças à compressão da fase de requisitos e da fase de implantação. Isso significa que, na obtenção dos requisitos, costumam-se optar por metodologias mais dinâmicas e rápidas, como workshops ao invés de entrevistas. Permite-se também um desenvolvimento

inicial no nível mais alto de abstração dos requisitos visto o envolvimento maior do usuário e visibilidade mais cedo dos protótipos (ver **Figura 5**).

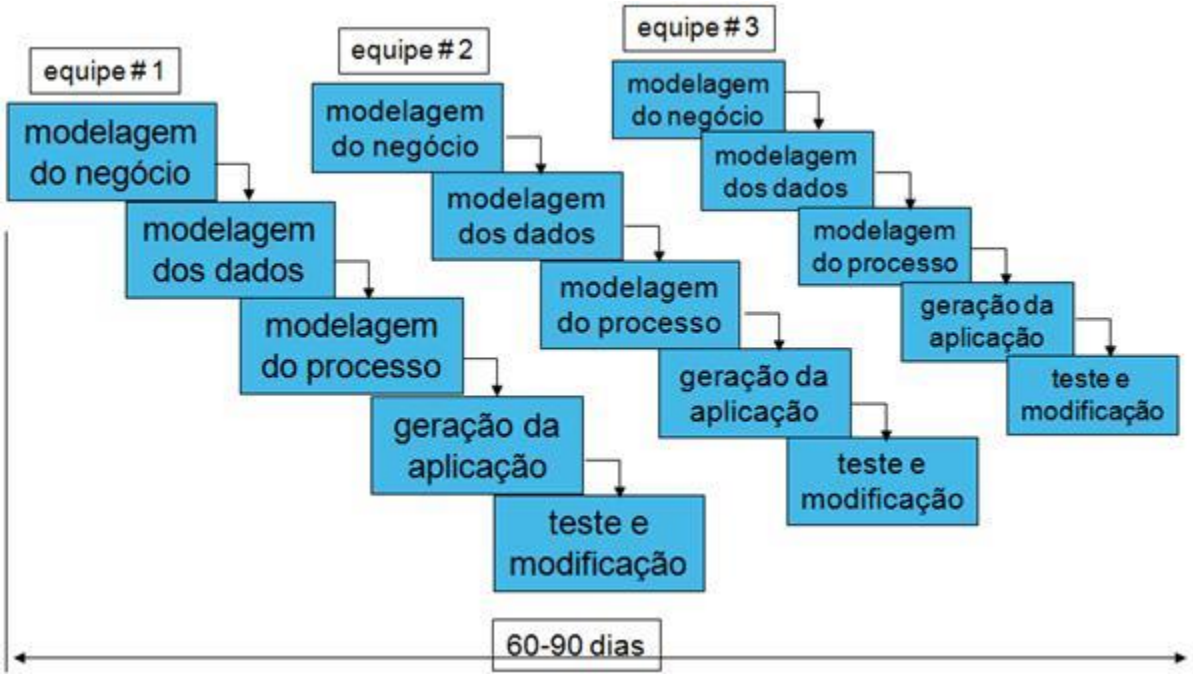


Figura 5. Ciclo de vida RAD

As fábricas de software que resolvem por adotar este modelo devem ter uma estrutura prévia diferencial de pessoas e ferramentas, tais como:

- Pessoas:
  - Profissionais experientes (funcional e gerência);
  - Profissionais de rápida adaptação;
  - Equipes de colaboração mútua;
  - Maior quantidade de pessoas;
- Gerenciamento:
  - Empresas pouco burocráticas que encorajem a eliminação de obstáculos;
  - Alto controle do tempo;
- Uso de Ferramentas:
  - CASE;
  - Muita diagramação;
  - Prévia biblioteca de componentes reutilizáveis (APIs, wizards, templates,...);
  - Fácil manutenção (ex.: linguagens de programação que suportem Orientação a Objetos, tratamento de exceção, ponteiros);
  - Adoção de ferramentas maduras, pois não há tempo de atualizar versões e tratar erros inesperados;

Os sistemas desenvolvidos no ciclo RAD tendem a ter uma padronização de telas muito forte, devido a bibliotecas reutilizáveis e templates, porém tendem a perder em desempenho do sistema e na análise de risco (atividades estas que demandam tempo em qualquer projeto). Assim, é preferível seu uso para softwares de distribuição pequena.

Prototipagem

Prototipagem é a construção de um exemplar do que foi entendido dos requisitos capturados do cliente. Pode ser considerado um ciclo de vida ou pode ser usado como ferramenta em outros ciclos de vida.

Um protótipo em engenharia de software pode ser o desenho de uma tela, um software contendo algumas funcionalidades do sistema. São considerados operacionais (quando já podem ser utilizados pelo cliente no ambiente real, ou seja, em produção), ou não operacionais (não estão aptos para serem utilizados em produção). Os protótipos podem ser descartados, ou reaproveitados para evoluírem até a versão final.

No ciclo de vida de prototipagem, não é exigido um conhecimento aprofundado dos requisitos num primeiro momento. Isso é bastante útil quando os requisitos não são totalmente conhecidos, são muitos complexos ou confusos. Desta forma, se o cliente não sabe expressar o que deseja (o que ocorre bastante quando não é um sistema legado), a melhor maneira de evitar que se perca tempo e recursos com uma má interpretação é a construção de modelos, ou seja, de protótipos do que o software faria.

Assim, o cliente experimentará, na prática, como o sistema ou parte dele funcionará. A partir desse primeiro contato, o cliente esclarece o que não foi bem interpretado, aprofunda alguns conceitos e até descobre um pouco mais sobre o que realmente precisa. A partir deste feedback, novos requisitos são colhidos e o projeto ganha maior profundidade. Outro protótipo é gerado e apresentado ao cliente, que retorna com mais feedbacks. Ou seja, o cliente participa ativamente do início ao fim do processo (ver **Figura 6**).

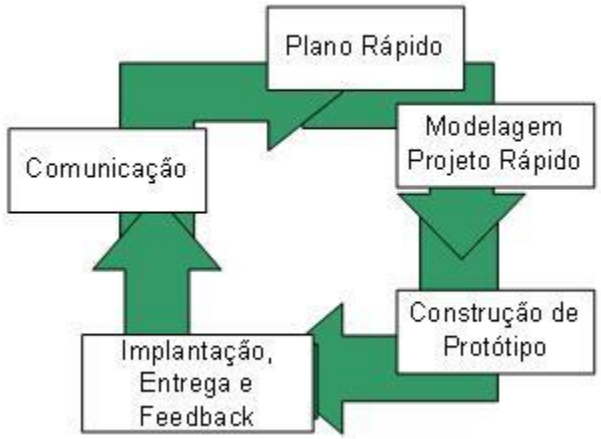


Figura 6. O modelo e prototipagem (Pressman, adaptado)

A geração de protótipos pode ser facilitada por ferramentas geradoras de telas, de relatórios, poupando esforço de programação e diminuindo o tempo de entrega.

Cada protótipo tem uma finalidade diferente. Um protótipo pode servir para esclarecer dúvidas sobre uma rotina, demonstrar a aparência das telas, conteúdo de tabelas, formato de relatórios. Os protótipos podem também ser utilizados para apresentar opções ao cliente para que ele escolha a que mais lhe agrade, como opções de navegação, de fluxo de telas, entre outras.

Por isso, é muito importante explicar previamente ao cliente que protótipos são apenas modelos para melhorar a comunicação. Caso contrário, pode causar uma frustração por não funcionar corretamente, ter funções limitadas, ter resposta lenta, ou a aparência ruim. Certamente um protótipo construído para esclarecer uma rotina provavelmente terá uma “cara feia”; para demonstrar a aparência das telas, não terá funcionalidade; para apresentar o formato dos relatórios, os dados não serão coerentes.

O cliente fará comparações entre o sistema final e o que foi “prometido” através do protótipo e pode ficar insatisfeito. Por exemplo, geralmente o protótipo não acessa rede ou banco de dados, pois as informações são “desenhadas” com a tela, fazendo com que tudo fique muito rápido. Já no ambiente operacional haverá uma degradação de desempenho e o cliente pode se decepcionar.

Faz parte de um bom gerenciamento no modelo de prototipagem planejar se, quais e que funções dos protótipos não operacionais serão reaproveitadas na versão operacional, para que sua confecção siga as **boas práticas de engenharia de software**. Os protótipos não operacionais são construídos com pouca qualidade em prol da velocidade. Ou seja, não há preocupação na programação, em refinar o código, em usar comentários, em aproveitar eficientemente os recursos de hardware e software, na manutenção, no reuso de componentes

e na integração com outras funções ou sistemas. Com certeza será um problema se a equipe sucumbir à pressão do cliente, cada vez mais ansioso para ver a versão final daquele trabalho, e transformar à revelia, protótipos não operacionais em operacionais.

O gerente também deve se preocupar com o escopo do projeto versus a quantidade de protótipos, para que não se perca muito tempo nesse processo, tampouco se transforme num processo de “tentativa e erro”.

Não é uma tarefa fácil documentar o modelo de ciclo de vida baseado na prototipagem devido aos requisitos não serem totalmente conhecidos no primeiro momento e a consequente quantidade de mudanças ocorridas.

Modelo Espiral

O modelo proposto por Boehm em 1988 trata de uma abordagem cíclica das fases do processo, onde a cada “volta” ou iteração temos versões evolucionárias do sistema.

Este é um modelo guiado por risco, suporta sistemas complexos e/ou de grande porte, onde falhas não são toleráveis. Para isso, a cada iteração há uma atividade dedicada à análise de riscos e apoiada através de geração de protótipos, não necessariamente operacionais (desenhos de tela, por exemplo) para que haja um envolvimento constante do cliente nas decisões.

Cada iteração ou volta é dedicada a uma fase do processo de vida de um software (viabilidade do projeto, definição de requisitos, desenvolvimento e teste,...). Ao mesmo tempo, cada volta é seccionada em 4 setores, da seguinte forma:

1. Iteração: Viabilidade do projeto:
    - 1.1. Definição de objetivos;
    - 1.2. Avaliação e redução de riscos;
    - 1.3. Desenvolvimento e validação;
    - 1.4. Planejamento da próxima fase;
2. Iteração: Definição de requisitos do sistema:
    - 2.1. Definição dos objetivos;
    - 2.2. Avaliação e redução de riscos;
    - 2.3. Desenvolvimento e validação;
    - 2.4. Planejamento da próxima fase;
3. Iteração: Projeto do sistema:
    - 3.1. ...
    - 3.2. ...
    - 3.3. ...
    - 3.4. ...
4. Iteração: Desenvolvimento e teste de unidade
    - 4.1. ...
    - 4.2. ...
    - ...
5. Iteração: Implantação
    - ...

Ou, na representação gráfica deste modelo conforme **Figura 7**.

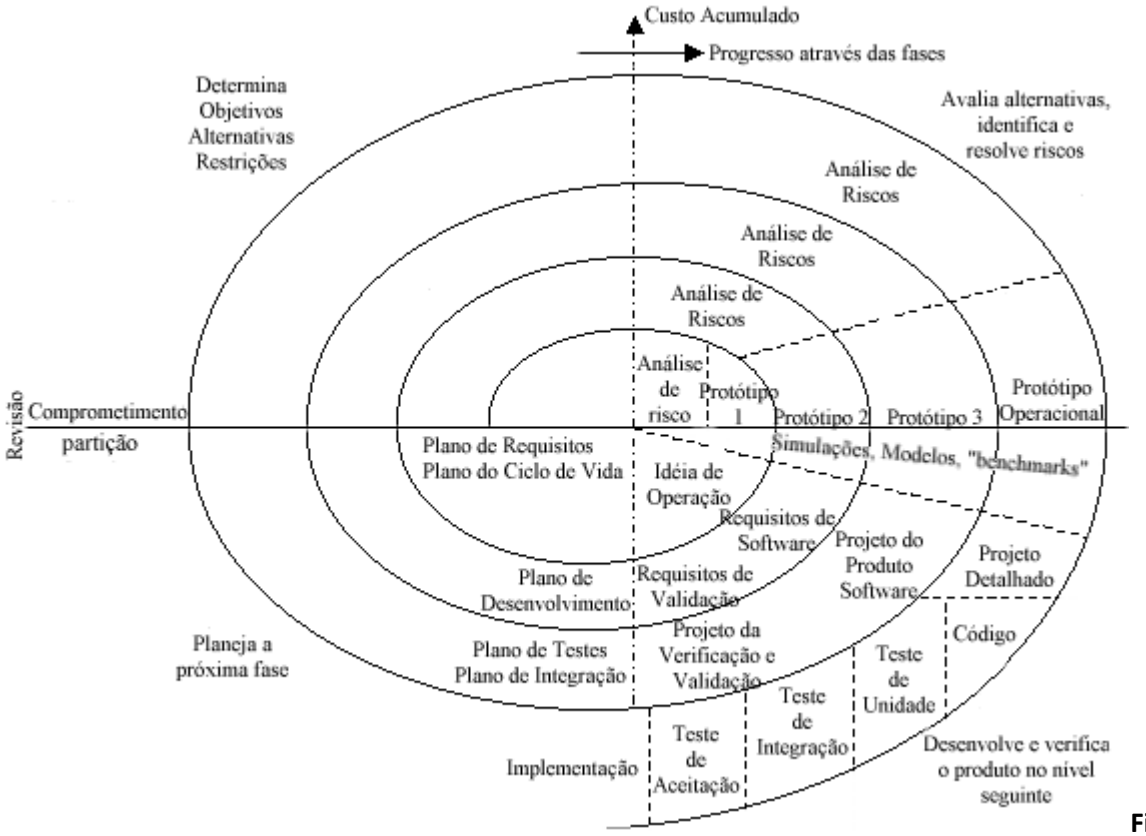


Figura 7. Modelo Espiral

Os quatro setores são explicados da seguinte forma:

1. Na Definição de Objetivos, desempenhos, funcionalidade, entre outros objetivos, são levantados. Visando alcançar esses objetivos são listadas alternativas e restrições, e cria-se um plano gerencial detalhado.
2. Na Análise de Riscos, as alternativas, restrições e riscos anteriormente levantados são avaliados. Neste setor (porém não apenas neste) protótipos são utilizados para ajudar na análise de riscos.
3. No Desenvolvimento e Validação um modelo apropriado para o desenvolvimento do sistema é escolhido, de acordo com o risco analisado no setor anterior (cascata, interativo,...).
4. No Planejamento da Próxima fase ocorre a revisão do projeto e a decisão de partir para a próxima fase.

Ou seja, cada volta ou iteração do processo é vista por quatro ângulos.

No final da Viabilidade do Projeto teremos como resultado a Concepção das Operações; da Definição de Requisitos o produto serão os requisitos; no final do Desenvolvimento e Testes o projeto é criado e os testes habilitados. Pode-se parar por aí, pode-se incluir mais fases, pode a espiral ficar adormecida até uma nova alteração do sistema se requisitada, e desta forma estender até o fim de vida do sistema.

Neste modelo, apenas o início é definido. A evolução e amadurecimento dos requisitos demandam tempo ajustável (assim como custo). Isto torna o sistema difícil de ser vender ao cliente e exige um alto nível de gerenciamento em todo o processo.

Modelo de Ciclo de Vida Associado ao RUP

Derivado **da UML** e do Processo Unificado de Desenvolvimento de Software, o RUP, Rational Unified Process, é um modelo de processo iterativo e incremental, dividido em fases, orientado a casos de uso. Possui framework (esqueleto) de processo e manuais que guiam na utilização das melhores práticas de especificação de projeto (Vídeo Aula sobre Ciclo de Vida de Software, parte 3, revista Engenharia de Software Magazine).

O objetivo do RUP é produzir software com qualidade (melhores práticas de engenharia de software) que satisfaça as necessidades dos clientes dentro de um prazo e orçamento estabelecidos.

Este modelo foi desenvolvido pela Rational Software Corporation e adquirido pela IBM, que o define da seguinte maneira: “IBM Rational Unified Process®, ou RUP, é uma plataforma de processo de desenvolvimento de software configurável que oferece melhores práticas comprovadas e uma arquitetura configurável. (ver **Figura 8**)”

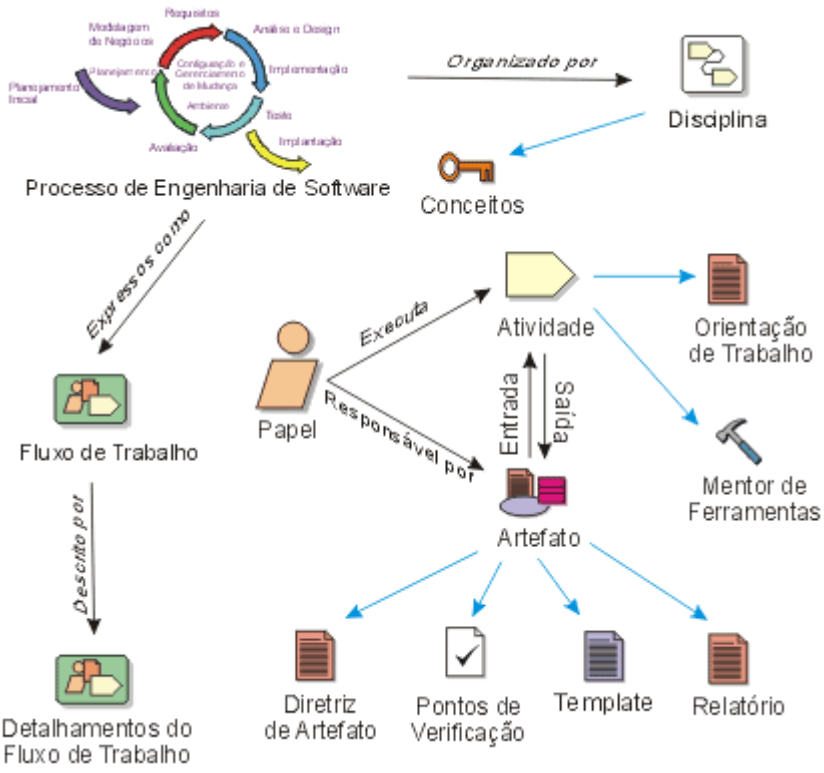


Figura 8. Conceitos chaves do RUP

O RUP possui quatro fases de negócio. O nome de cada fase revela o que será entregue por ela (ver **Figura 9**):

- Concepção: define o escopo do projeto, ou “business case”; onde é julgado se o projeto deve ir adiante ou ser cancelado.
- Elaboração: elabora **modelo de requisitos**, arquitetura do sistema, plano de **desenvolvimento para o software** e identificar os riscos.
- Construção: constrói o software e a documentação associada.
- Transição: finaliza produto, define-se plano de entrega e entrega a versão operacional documentada para o cliente.

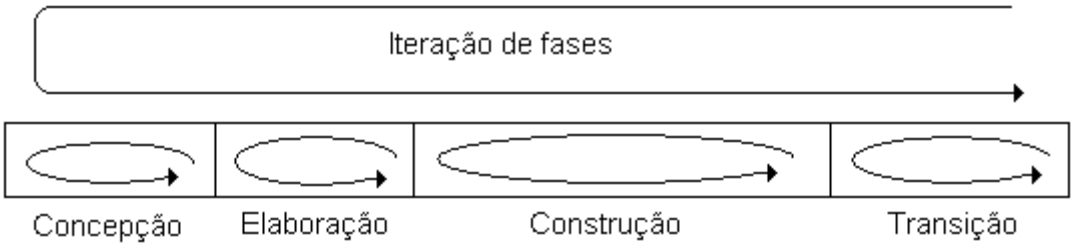


Figura 9. RUP

A iteração no RUP tem por objetivo minimizar os riscos. Como pode ser visto na **Figura 9**, a iteração pode acontecer dentro de cada fase, gerando incrementos, ou em todo o processo. Por exemplo, dentro da concepção, a iteração pode ocorrer até que todos os requisitos sejam perfeitamente entendidos. O plano de iterações identificará quais e quantas iterações são necessárias durante o processo.

Em geral, essas fases demandam esforço e programação diferentes. Para um projeto de médio porte, de acordo com o fabricante será seguida a distribuição apresentada na **Tabela 1**.

	Concepção	Elaboração	Construção	Transição
Esforço	~5%	20%	65%	10%
Programação	10%	30%	50%	10%

Tabela 1. Distribuição prevista de esforço e programação para um ciclo de desenvolvimento inicial típico de um projeto de médio tamanho

O RUP usa templates que descrevem o que é esperado no resultado de cada fase ou cada iteração (IBM, 2004), identificando as competências e responsabilidades (arquiteto, analista, testador,...), as atividades e os artefatos.

Para descrever as atividades (codificação de uma classe, integração de sistemas,...) o RUP faz o uso de manuais (guidelines), que descrevem técnicas e heurísticas; e de "Mentores de Ferramentas", que explicam o uso da ferramenta para executar a atividade. Os artefatos de cada fase (documentos, modelos, códigos, etc.) são criados, juntamente com templates e exemplos, para melhor entendimento da equipe e do cliente (ver **Figura 10**).

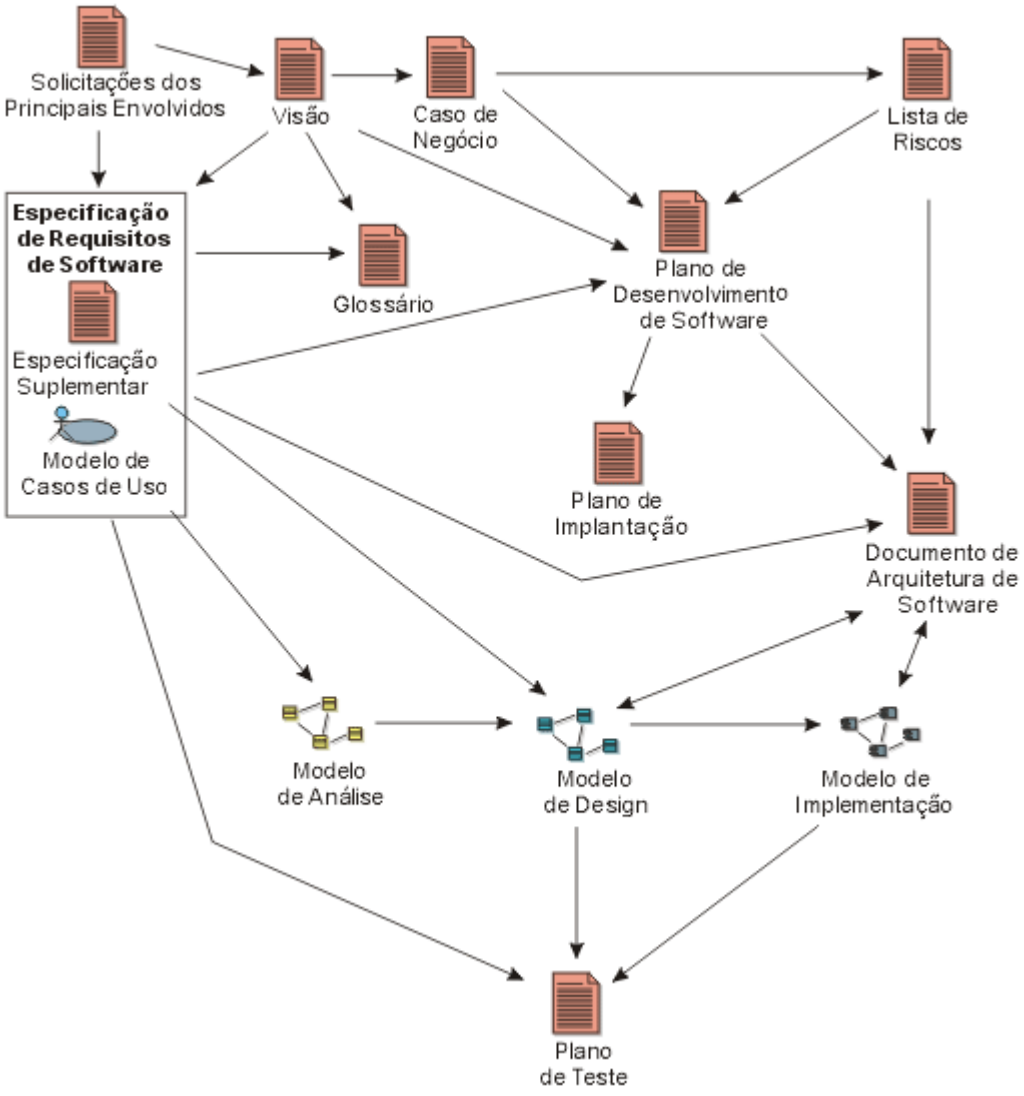


Figura 10. Os principais artefatos do Rational Unified Process e o fluxo de informações existente entre eles

Os templates também ajudam no gerenciamento, pois definem o que precisa ser executado. Servem também como guia para que as boas práticas de especificação de projeto não sejam esquecidas no processo de desenvolvimento daquele software.

Assim, toda a preocupação dada pelo RUP em disciplinar o processo através de frameworks, guias, templates, faz com que haja uma melhor alocação de pessoas na equipe, padronização do sistema, visão concreta do andamento do projeto.

A escolha do RUP deve ser feita por empresas de software com prévia experiência, pois a definição de framework, templates, guias, métodos, entre outros, demandam tempo e exigem aderência às boas práticas de processo de software.

Considerações Finais

Finalizando este artigo sobre os **modelos de ciclo de vida de software**, segue uma tabela comparativa das principais características que devem ser observadas antes de escolher o ciclo ou os ciclos de vida a serem adotados (ver **Tabela 2**).

Vale ressaltar que, conforme já mencionado anteriormente, não existe um modelo ideal e na maioria dos softwares desenvolvidos são utilizados mais de um modelo de ciclo de vida.

Modelo	Foco	Requisitos	1ª versão p/ cliente	Gerenciamento (1=mais simples)	Sistemas (tamanhocomplexidademáximos)
Cascata	Documento e artefato	Bem conhecido/congelado	Fim do ciclo	1	Simples
V	Planejamento de testes	Bem conhecido/congelado	Fim do ciclo	2	Simples
Incremental	Incrementos operacionais	Maior abstração / Tratado em módulos	Protótipos operacionais	3	Médio
Evolutivo	Evolução dos requisitos	Pouco conhecidos	Protótipos operacionais	4	Médio
RAD	Rapidez	Escopo restrito / Maior abstração /	Protótipos operacionais	4	Médio

		Tratado em módulos			
Prototipagem	Dúvidas nos Requisitos	Abstratos	Protótipos não operacionais	5	Médio
Espiral	Análise de risco	Maior abstração / evoluídos com o tempo	Protótipos operacionais ou não operacionais	5	Complexos
RUP	Frameworks e boas práticas	Maior abstração / evoluídos com o tempo	Protótipos operacionais ou não operacionais	5	Complexos

Tabela 2. Comparação entre os modelos de ciclo de vida

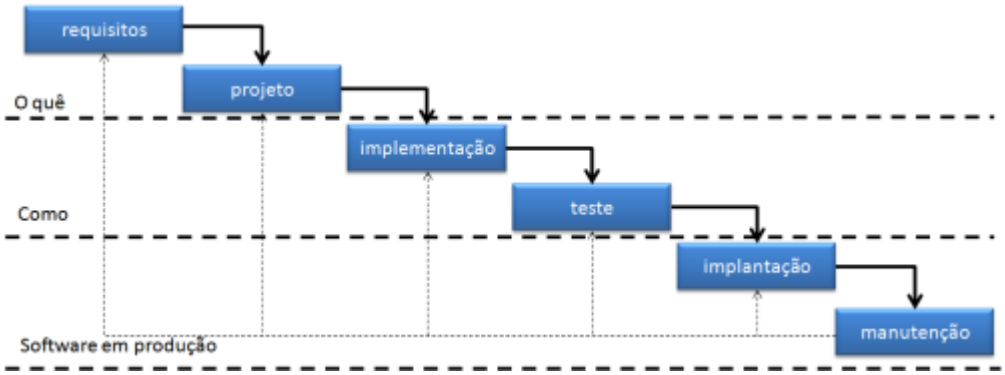
Ref. https://www.devmedia.com.br/ciclos-de-vida-do-software/21099

Modelos de Ciclo de Vida de Software

Nesse post vou abordar alguns dos modelos de ciclo de vida de desenvolvimento de software. Para escrever esse conteúdo estou utilizando as aulas disponibilizadas por esse site http://cederj.edu.br/videoaulas/ além de outros conteúdos que pesquisei pela internet. 27 de janeiro de 2014 Adonai Canêz

1) Modelo em Cascata/Clássico (Waterfall Model)

* Popularizado na década de 1970;
* Composto por uma sequência de atividades;
* Uma atividade começa a executar quando a outra termina;
* Resultado de uma etapa é utilizado na etapa seguinte;
* Guiado por documentos;
* Ciclo de vida mais antigo e mais utilizado.



Modelo em Cascata

a) Vantagens:

* Oferece uma maneira de tornar o processo mais visível;
* Facilita o planejamento.

b) Desvantagens:

* Dificuldade de manter a serialização proposta pelo modelo
* Dificuldade de se concluir a etapa de análise de requisitos, devido a modificações nos requisitos do software (requisitos deveriam ser congelados ao fim da análise)
* A primeira versão do software só estará disponível após o término das fases de análise, projeto, codificação e testes, aumentando o tempo de latência entre o início do projeto e a criação de sua primeira versão.

2) Prototipação

* O objetivo é entender os requisitos do usuário;
* Versão simplificada de um produto de software, geralmente criada sem um processo formal de desenvolvimento, utilizada para elucida ou validar os requisitos do produto.

a) Vantagens:

* Protótipos contribuem para melhorar a especificação dos requisitos;
* Partes do protótipo podem vir a ser usadas no desenvolvimento do sistema final.

b) Desvantagens:

* Custo elevado;
* Atraso no desenvolvimento.

3) Modelo Incremental

* Combinação entre os modelos linear e prototipação;
* O desenvolvimento é dividido em etapas, denominadas "incrementos";
* Em cada incremento é realizado todo o ciclo do desenvolvimento de software;
* Cada etapa produz um sistema totalmente funcional;

a) Vantagens:

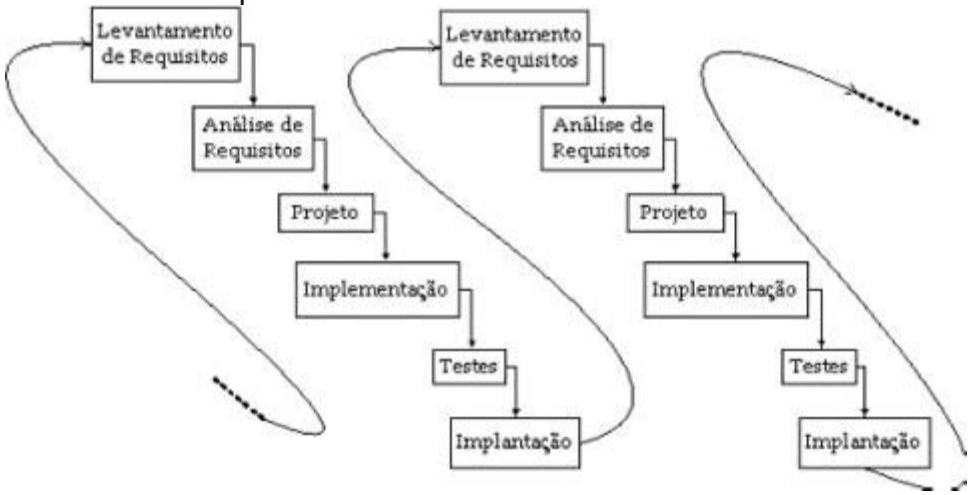
* Existe um risco menor de fracasso do software;
* Reduz a chance de mudança de requisito;

Tipos de Incrementos:

* Evolutivos: Produtos de cada etapa de desenvolvimento são aproveitados em cada nova passagem pela etapa.
* Descartáveis: Produtos das etapas de desenvolvimento são descartados a cada novo protótipo é construído do início.
* Operacional: Requisitos são elucidados através de protótipos e o produto final é construído paralelamente a construção dos protótipos.

3.1) Ciclo Incremental Evolutivo

* Os requisitos vão sendo implementados por parte
* Entregas rápidas de produtos intermediários para o usuário
* Maior entendimento a medida que o desenvolvimento ocorre



Ciclo Incremental Evolutivo

3.2) Ciclo Incremental Descartável

* Custo alto pela perda do produto construído no incremento

* Alta qualidade do processo por trás do protótipo descartável
* Possível insistência do usuário em manter o protótipo em uso.

3.3) Ciclo Incremental Operacional

* Mantem duas equipes

– Uma equipe pequena trabalha em turnos longos no ambiente do usuário, construindo protótipos "quick-&-dirty" para facilitar o entendimento dos requisitos do usuário.

– Uma equipe trabalha em ritmo menos acelerado e em um ambiente separado, produzindo um software de qualidade a partir dos requisitos coletados pela primeira equipe.

4) Ciclo de Vida em Espiral

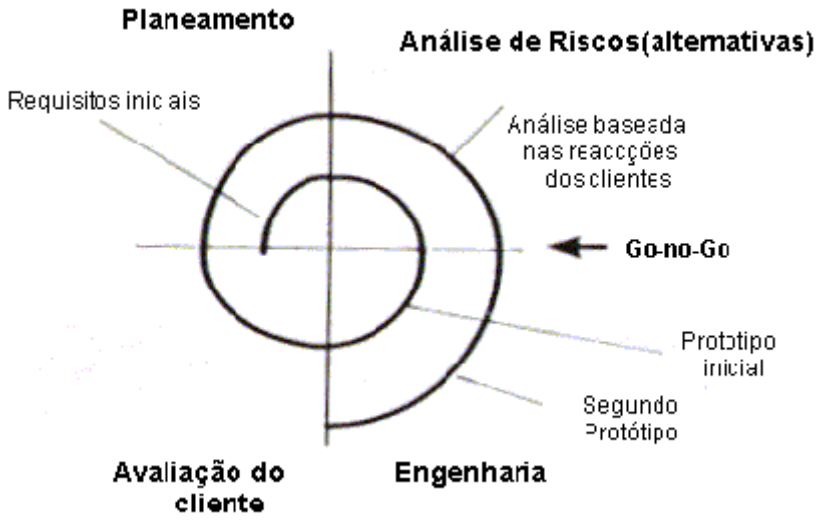
* É um metamodelo, visto que qualquer ciclo de vida pode ser utilizado na fase de desenvolvimento.
* Foi criado visando abranger as melhores características do modelo clássico e da prototipagem;
* Dividido em quatro fases;
* Essas fases são repetidas várias vezes;

a) Vantagens:

* Possibilidade de melhorar o sistema a cada iteração;
* Diminui manutenção.

b) Desvantagens:

* Modelo relativamente novo;
* Modelo mais complexo.



Modelo em Espiral

5) Ciclo de Vida RAD (Rapid Application Development)

* É um modelo que enfatiza um ciclo de desenvolvimento curto;
* Construção baseada em componentes;
* O modelo RAD é usado principalmente para aplicações de sistema de informação;

* Adaptação do modelo em cascata, onde um desenvolvimento rápido é obtido através de uma abordagem de construção baseada em componentes.

a) Vantagens:

* Baseado em componentes;
* Pode ser desenvolvido em várias equipes;

b) Desvantagens:

* É necessário desenvolvedor e cliente comprometidos;
* Não é apropriado para qualquer aplicação;

Critérios para Seleção de um Ciclo de Vida

* Relacionados aos usuários e a equipe
– Experiencia dos usuários no domínio da aplicação;
– Facilidade de expressão dos usuários;
– Experiencia da equipe no domínio da aplicação;
– Experiencia da equipe de Engenharia de Software;
– Disponibilidade de recursos humanos para a equipe;
– Grau de acesso aos usuários.

* Relacionados ao Produto
– Tamanho e grau de complexidade da aplicação
– Grau de importância dos requisitos de interface

Ref. https://www.adonai.eti.br/2014/01/modelos-de-ciclo-de-vida-de-software/