O **ciclo de vida de software** corresponde ao **conjunto de etapas** pelas quais um sistema passa, desde a concepção até sua desativação. Ele serve como um guia estruturado que orienta as equipes no(a):

- **análise,**
- **projeto,**
- **desenvolvimento,**
- **teste,**
- **implantação e manutenção do produto**,

garantindo organização e qualidade em cada fase.

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.”

Existem diferentes modelos de ciclo de vida, sendo o **modelo em Cascata (Waterfall)** e o **modelo Ágil (Agile)** dois dos mais conhecidos. Enquanto o primeiro segue uma sequência linear e rígida de fases, o segundo prioriza a flexibilidade, a adaptação contínua e a entrega incremental de valor.

Em essência, o ciclo de vida de software representa a estrutura de processos, atividades e tarefas relacionadas ao desenvolvimento, operação e manutenção de um sistema, abrangendo desde a definição de requisitos até o encerramento de seu uso.

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.

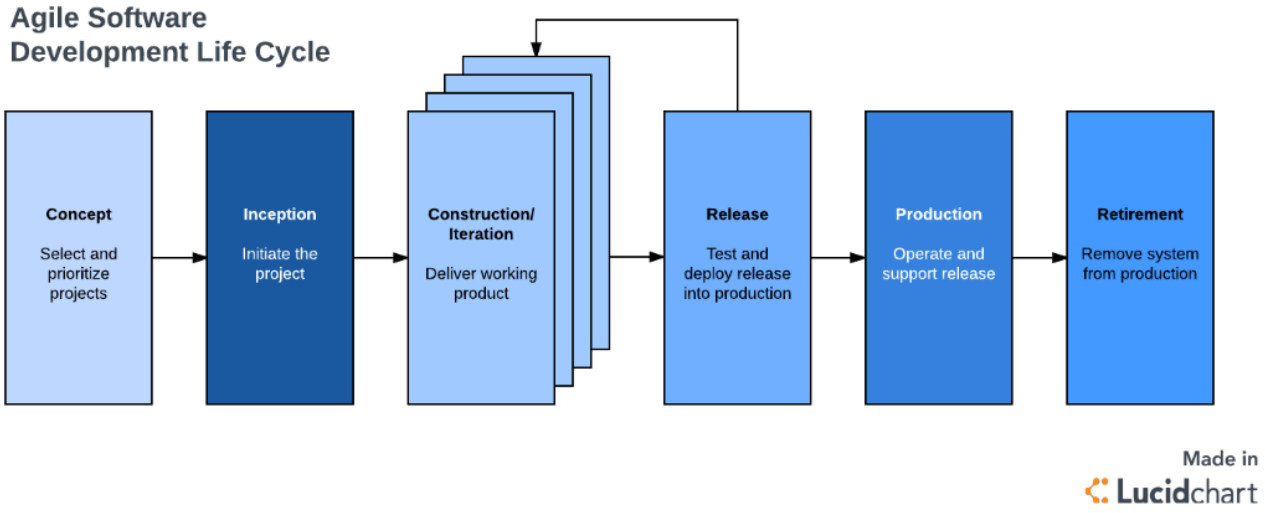
Ciclo de desenvolvimento Ágil de software

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. Visão gral do ciclo de vida de desenvolvimento de software ágil.



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.

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.

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.

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

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**.

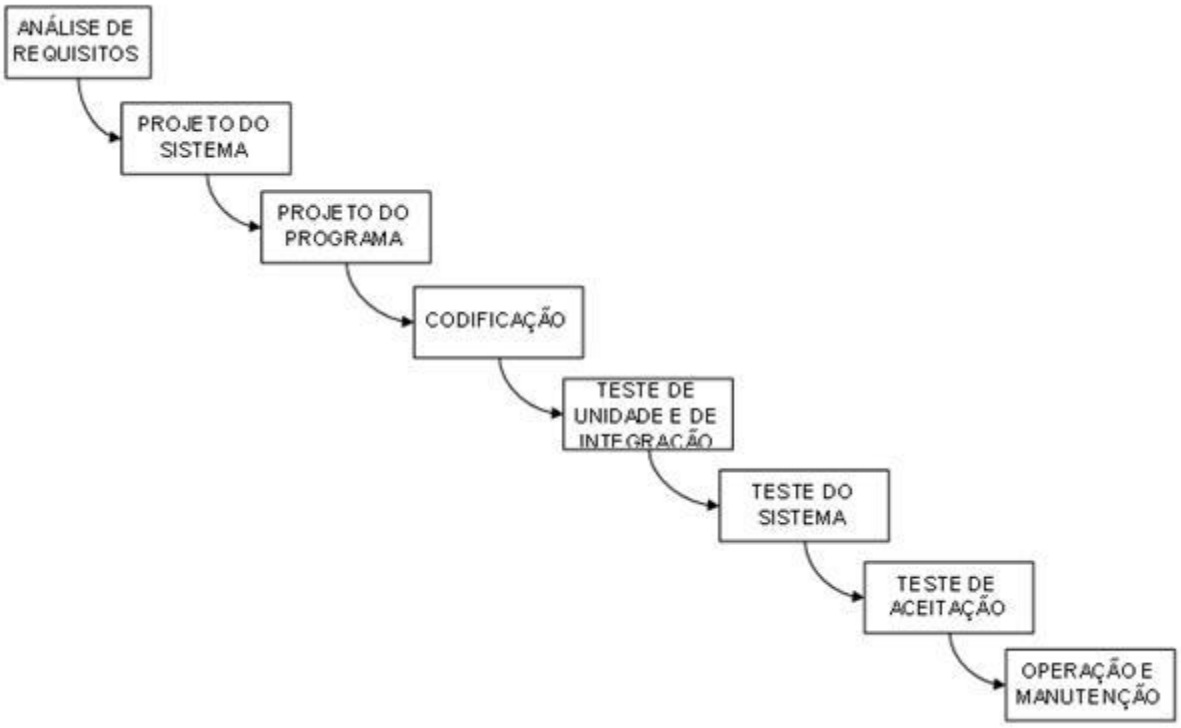


Figura 1. O modelo em cascata

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.

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**).

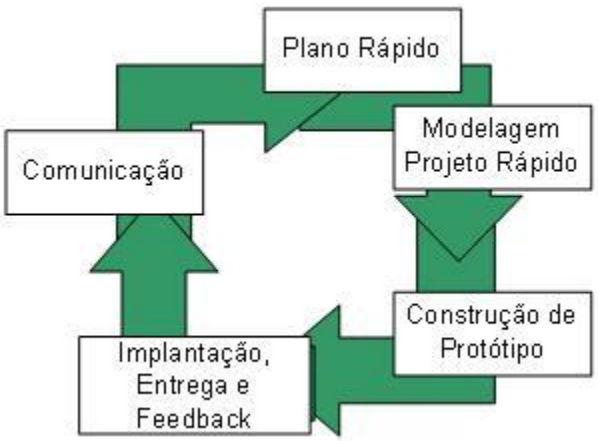


Figura 6. O modelo e prototipagem (Pressman, adaptado)

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.

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.

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**)"

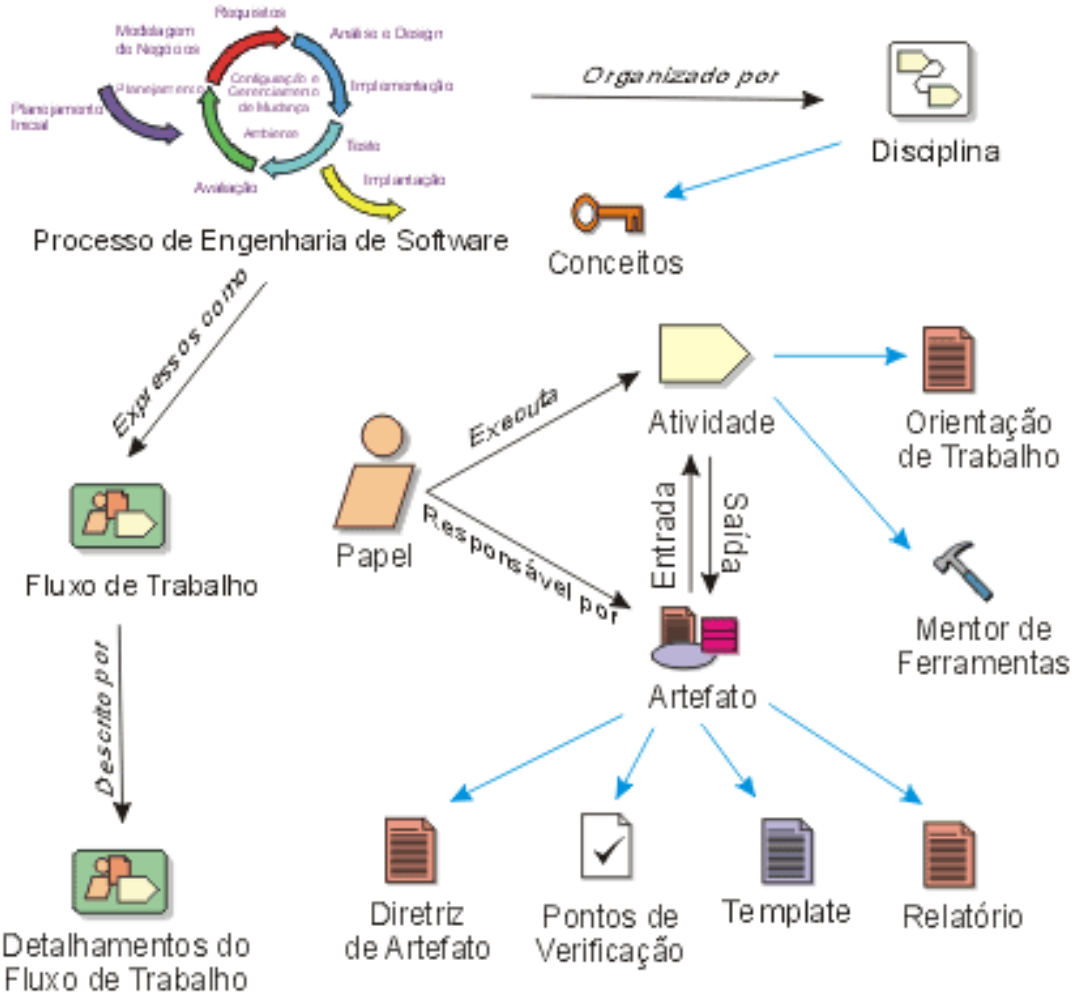


Figura 8. Conceitos chaves do RUP

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.

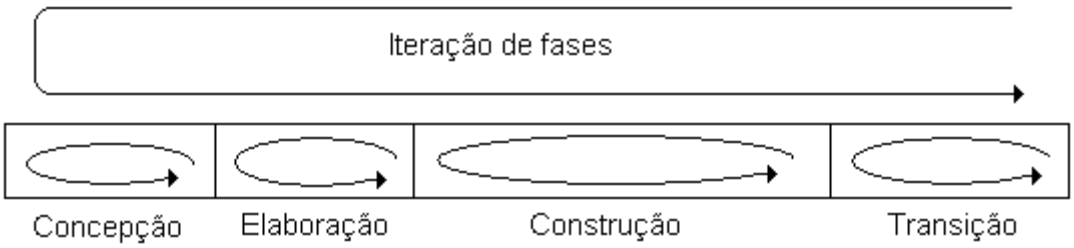


Figura 9. RUP

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.

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.

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