



EBSERH
HOSPITAIS UNIVERSITÁRIOS FEDERAIS

Identificação geral

Nome	**Empresa Brasileira de Serviços Hospitalares**
CNPJ	15.126.437/0001-43
Sede	Brasília-DF
Tipo de estatal	Empresa Pública
Acionista controlador	União
Tipo societário	Sociedade por Cotas de Responsabilidade Limitada
Tipo de capital	Fechado
Abrangência de atuação	Nacional
Setores de atuação	Educação e Saúde
Presidente	**Arthur Chioro** Telefone: (61) 3255-8537 E-mail: chefiadegabinete.sede@ebserh.gov.br
Comitê Gestor de Tecnologia da Informação e Comunicações (CGTIC)	**Daniel Gomes Monteiro Beltrammi** Cargo: Vice-Presidente **Giliate Cardoso Coelho Neto** Cargo: Diretor de Tecnologia da Informação **Iara Ferreira Pinheiro** Cargo: Diretora de Orçamento e Finanças **Lumena Almeida Castro Furtado** Cargo: Diretora de Ensino, Pesquisa e Atenção à Saúde **Odete Carmen Gialdi** Cargo: Diretora de Administração e Infraestrutura **Cristiane Carvalho Santos Melo** Cargo: Diretora de Gestão de Pessoas
Equipe de elaboração da Metodologia de Desenvolvimento de Sistemas	**André Gomes Alay Esteves** **André Luiz de Noronha** **Clayr Madeira de Albuquerque Silva** **Daniel da Silva Farias** **Diego Souza Silva Almeida** **Eliane Cunha Marques** **Hilton Pinheiro Mendes Sobrinho** **Igor de Andrade Marrocos** **Jailson Pereira Oliveira** **Kelvis Lima Almeida** **Leandro Noronha do Nascimento** **Luanna Siqueira de Assis** **Luciano Nunes Guedes** **Paulo Cesar Guimaraes Campos** **Rodrigo de Souza Rezende** **Rodrigo Vaz dos Santos** **Telmo Nunes Costa** **Wilson Miranda Clementino**
Data da Divulgação	Março/2023
Versão	5.2

Sumário

1. Introdução

A Metodologia Desenvolvimento de Sistemas, Sustentação e Implantação de Sistemas da Empresa Brasileira de Serviços Hospitalares (MDS-Ebserh) define os processos e atividades dos serviços de desenvolvimento, sustentação e implantação de aplicações de *software* no âmbito da Diretoria de Tecnologia da Informação (DTI), apresentando o detalhamento das fases do ciclo de vida, descrição de atividades, papéis, responsabilidades e produtos gerados em cada etapa.

2. Diretrizes

A MDS-Ebserh é um guia baseado em diversas abordagens e por isso é considerada uma metodologia híbrida, integrando valores e métodos ágeis com boas práticas de engenharia de *software*, gerenciamento de projetos e governança de Tecnologia da Informação (TI), priorizando as metodologias e *frameworks* ágeis como *Scrum* e Lean-IT, bem como princípios e práticas da integração entre os processos de desenvolvimento e operação por meio de ferramentas de automação de integração contínua, testes, entrega contínua e gerenciamento de infraestrutura (DevOps).

3. Justificativa

Visando padronizar e institucionalizar os processos de desenvolvimento, sustentação e implantação de aplicações de *software* da Diretoria de Tecnologia da Informação (DTI), fez-se necessária a definição de regras e padrões que orquestrem as ações estruturantes que viabilizem à DTI cumprir sua missão institucional de forma alinhada com as diretrizes da Administração Pública e com as melhores práticas de mercado no desenvolvimento, na sustentação e na implantação de sistemas de informação.

4. Objetivo

Padronizar os processos de trabalho, as funções desempenhadas e as tarefas a serem realizadas por todos os envolvidos no processo de desenvolvimento, sustentação e implantação de aplicações de *software* da Diretoria de Tecnologia da Informação (DTI).

5. Aplicabilidade

As diretrizes, processos e atividades estabelecidos nesta metodologia são aplicáveis a todos os sistemas desenvolvidos e/ou mantidos pela DTI, promovendo, assim, padronização, eficiência, eficácia e efetividade do processo de desenvolvimento, sustentação e implantação de aplicações de *software*.

6. Política de Revisão

Este documento poderá ser atualizado pela DTI sempre que houver necessidade de atualização, seja por motivo de adequação aos normativos institucionais, diretrizes gerenciais ou

recomendações de órgãos de controle, bem como revisão do texto para eliminar eventuais ambiguidades, omissões ou contradições.

7. Conceitos

Desenvolvimento: compreende as atividades de desenvolvimento e evolução (manutenções evolutivas, perfectivas, adaptativas e integrativas) de aplicações de *software* em plataformas web, desktop e mobile, que englobam a análise e o levantamento de requisitos, a construção e atualização de artefatos de documentação e do código-fonte da aplicação, bem como a execução de testes funcionais.

Sustentação: compreende as atividades realizadas para a manutenção da disponibilidade, estabilidade e desempenho das aplicações de *software*, tais como manutenções corretivas, investigação de incidentes, resolução de problemas e apoio aos usuários, bem como atividades de suporte à manipulação de dados, monitoramento e operação das aplicações e seus componentes, compreendendo todos os ambientes em que a solução esteja instalada.

Implantação: compreende as atividades de preparação de ambientes de aplicação de *software*, além das atividades de inserção desse sistema no rol soluções de aplicações, tais como planejamento, parametrizações, configurações, treinamentos técnicos, funcionais e operacionais, além de apoio na intermediação das evoluções, apoio no endomarketing e na elaboração de materiais de apoio à utilização do sistema (manuais, vídeos, orientações etc.).

8. Processo de Desenvolvimento

Para projetos de desenvolvimento de aplicações de *software*, será utilizado o framework *Scrum* como referência de práticas ágeis.

Pilares

Transparência: aspectos significativos do processo devem estar visíveis aos responsáveis pelos resultados. Esta transparência requer aspectos definidos por um padrão comum para que os observadores compartilharem um mesmo entendimento do que está sendo visto;

Inspeção: os usuários *Scrum* devem, frequentemente, inspecionar os artefatos *Scrum* e o progresso em direção a detectar variações. Esta inspeção não deve, no entanto, ser tão frequente que atrapalhe a própria execução das tarefas. As inspeções são mais benéficas quando realizadas de forma diligente por inspetores especializados no trabalho a se verificar;

Adaptação: se um inspetor determina que um ou mais aspectos de um processo desviou para fora dos limites aceitáveis, e que o produto resultado será inaceitável, o processo ou o material sendo produzido deve ser ajustado. O ajuste deve ser realizado o mais breve possível para minimizar mais desvios.

Valores

Coragem: os integrantes de um projeto *Scrum* (*Scrum* Master + Product Owner + Time de Desenvolvimento) precisam ter coragem para fazer a coisa certa e trabalharem juntos removendo impedimentos, buscando soluções.

Foco: os integrantes de um projeto *Scrum* precisam focar no trabalho durante o Sprint e nas metas designadas pela empresa. Time disperso perde produtividade e não alcança os objetivos.

Comprometimento: cada integrante de um projeto *Scrum* se compromete com o trabalho que se responsabilizou em fazer. Se comprometer é se envolver e não abandonar o trabalho pela metade ou entregar sem qualidade.

Respeito: respeitar uns aos outros é o primeiro passo para manter a colaboração, a integração e bom ambiente de trabalho

Abertura: poder ser franco, expor as ideias e propostas mesmo que elas não sejam aproveitadas. Promover momentos de debates, discussões, ouvir opiniões e sugestões para gerar novas práticas.

Time *Scrum*

Product Owner (Dono do Produto): responsável por maximizar o valor do produto e do trabalho do Time de Desenvolvimento. É responsável também por gerenciar o Backlog do Produto.

Scrum Master: responsável por garantir que o *Scrum* seja entendido e aplicado. O *Scrum* Master faz isso para garantir que o Time *Scrum* possa aderir à teoria, práticas e regras do *Scrum*.

Time de Desenvolvimento: consiste em profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto "Pronto" ao final de cada Sprint.

O tamanho ideal do Time de Desenvolvimento é ser pequeno o suficiente para se manter ágil e grande o suficiente para completar um trabalho significativo dentro da Sprint. Menos de três integrantes no Time diminuem a interação e resultam em um menor ganho de produtividade. Times menores podem encontrar restrições de habilidades durante a Sprint, sendo incapaz de entregar um incremento potencialmente utilizável. Havendo mais de nove integrantes é exigida muita coordenação. Times grandes geram muita complexidade para que um processo empírico seja útil. Os papéis de Product Owner e de *Scrum* Master não são incluídos nesta contagem, a menos que eles também executem o trabalho do Backlog da Sprint.

O Time de Desenvolvimento deverá ser composto, no mínimo, pelos perfis profissionais *Scrum* Master, Analista de Requisitos e Engenheiro de *Software*. Demais perfis poderão compor o time, de acordo com a necessidade de cada projeto.

Eventos

Sprint: um time-boxed de uma a quatro semanas, durante o qual um "Pronto", versão incremental potencialmente utilizável do produto, é criado. Sprints tem durações coerentes em todo o esforço de desenvolvimento. Uma nova Sprint inicia imediatamente após a conclusão da Sprint anterior.

Planejamento de Sprint: reunião de planejamento da Sprint, na qual são definidos o objetivo, as entregas e o trabalho necessário para entregar o incremento da Sprint.

Reunião diária: é um evento time-boxed de 15 minutos, para que o Dev Team possa sincronizar as atividades e criar um plano para as próximas 24 horas. Esta reunião é feita para inspecionar o trabalho desde a última Reunião Diária, e prever o trabalho que deverá ser feito antes da próxima Reunião Diária.

Revisão da Sprint: reunião executada ao final da Sprint para inspecionar o incremento e adaptar o Backlog do Produto, se necessário. Durante a reunião de Revisão da Sprint o Time *Scrum* e as partes interessadas colaboram sobre o que foi feito na Sprint. Com base nisso e em qualquer mudança no Backlog do Produto durante a Sprint, os participantes colaboram nas próximas coisas que podem ser feitas para otimizar valor.

Retrospectiva da Sprint: a Retrospectiva da Sprint é uma oportunidade para o Time *Scrum* inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima Sprint. A reunião ocorre depois da Revisão da Sprint e antes da reunião de planejamento da próxima Sprint.

Artefatos

Backlog do Produto: é uma lista ordenada de tudo que deve ser necessário no produto, e é uma origem única dos requisitos para qualquer mudança a ser feita no produto. O Product Owner é responsável pelo Backlog do Produto, incluindo seu conteúdo, disponibilidade e ordenação.

Backlog da Sprint: é um conjunto de itens do Backlog do Produto selecionados para a Sprint, juntamente com o plano para entregar o incremento do produto e atingir o objetivo da Sprint. O Backlog da Sprint é a previsão do Time de Desenvolvimento sobre qual funcionalidade estará no próximo incremento e sobre o trabalho necessário para entregar essa funcionalidade em um incremento "Pronto".

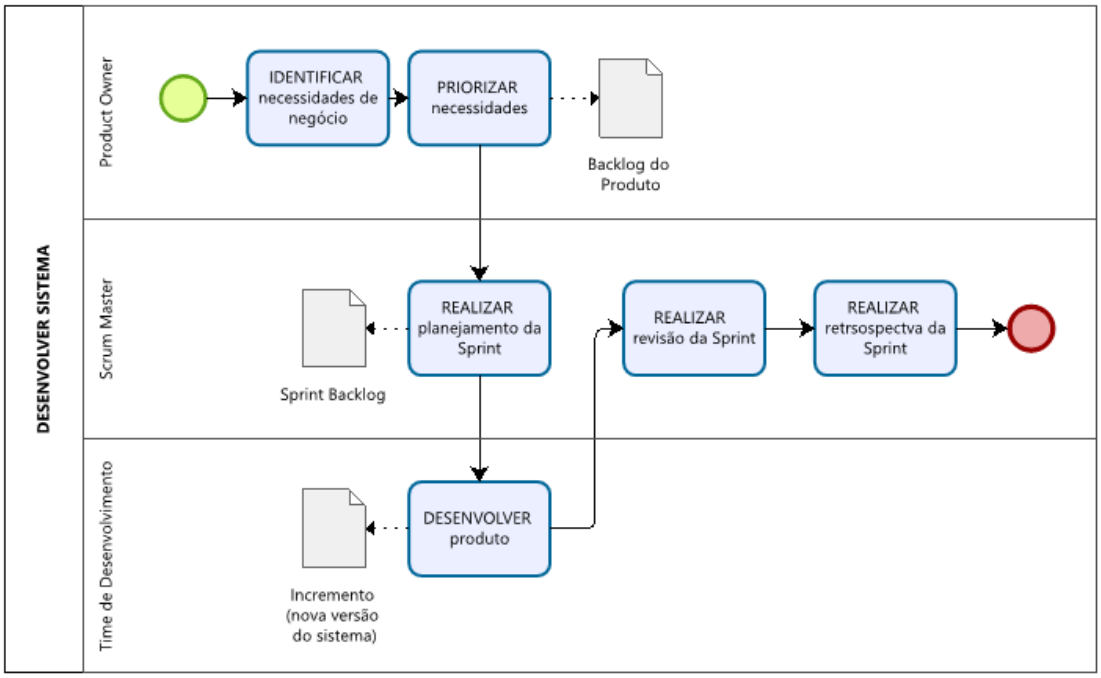
Incremento: é a soma de todos os itens do Backlog do Produto completados durante a Sprint e o valor dos incrementos de todas as Sprints anteriores. Ao final da Sprint um novo incremento deve estar "Pronto", o que significa que deve estar na condição utilizável e atender a definição de "Pronto" do Time *Scrum*. Este deve estar na condição utilizável independente do Product Owner decidir por liberá-lo realmente ou não.

NOTA: A última versão do framework *Scrum* não apresenta mais o conceito de Release. A documentação destaca que, embora Releases sejam comuns no desenvolvimento de *software*,

elas destacam os prontos fracos de desenvolvimento e não aproveitam o processo incremental e iterativo de framework ágil como o *Scrum*. Quando o *Scrum* é utilizado, um incremento potencialmente utilizável é entregue ao final de cada Sprint. Isso não significa que a entrega realizada é útil para os usuários. Portanto, muitas vezes há necessidade de lotes de incremento utilizáveis em Sprint subsequentes para que algo potencialmente útil seja entregue aos usuários. Infelizmente, algumas organizações usam as Releases para compensar as más práticas de desenvolvimento. Um incremento utilizável é aquele sem nenhum trabalho "não feito" associado a ele no final de um Sprint. Se as equipes não conseguirem criar incrementos realmente utilizáveis, o trabalho restante (teste, documentação, aceitação do usuário etc.) será feito como parte do processo de Release (às vezes chamado de fase de estabilização), potencializando a aceitação das más práticas de desenvolvimento.

Artefato		**Novo projeto**			**Projeto de melhoria**		
		Obrigatório	Opcional	Não se aplica	Obrigatório	Opcional	Não se aplica
Backlog do Produto		X	-	-	X	-	-
Backlog da Sprint		X	-	-	X	-	-
Incremento	História de usuário	X	-	-	X	-	-
	Código-fonte	X	-	-	X	-	-
	Script de banco de dados	X	-	-	X	-	-
	Manual de usuário	X	-	-	-	X	-
	Notas de release	X	-	-	X	-	-

Processo de Desenvolvimento



9. Processo de Sustentação

Para a sustentação de aplicações de *software*, serão utilizados princípios Lean-IT como referência para entrega de valor. O ponto central do Lean-IT é eliminar o desperdício, onde desperdício é entendido como trabalho que não adiciona valor ao produto ou serviço.

Tipos de desperdício

Tipo de desperdício	Exemplo
Defeito	• Incidentes e problemas que geram interrupções em aplicações, infraestrutura e serviços • Mudanças com falha que geram interrupções na produção • Liberação de atualizações que contém erros de programação (bugs) • Produção de dados imprecisos
Produção em excesso (superdimensionamento)	• Operar com capacidade de processamento, rede, armazenamento e/ou memória maior que o que realmente é necessário ou atualmente utilizado • Relatórios, dashboards, catálogos de serviço e portais que fornecem muito mais informações do que é realmente necessário
Espera	• Tempo de resposta da aplicação alto • Atrasos causados pelo escalonamento das requisições para outras equipes de suporte • Espera pelo atendimento de um analista da Central de Serviços em uma situação que poderia facilmente ser feita por um recurso de autoajuda
Excesso de processamento	• Criação de relatórios técnicos para áreas de negócio
Transporte	• Vistas presenciais para resolver problemas de hardware e/ou *software*
Inventário (excesso)	• Produção de sistemas que não são utilizados
Movimento (excesso)	• Viagens para comparecer em reuniões que podem ser realizadas virtualmente, sem a necessidade de deslocamento
Conhecimento (falta de uso)	• Problemas na retenção de conhecimento • Gasto de tempo em tarefas repetitivas

Princípios

Em TI, os fluxos de valores são os serviços fornecidos pela área de Tecnologia da Informação para a organização e para o uso de clientes, empregados, órgãos reguladores e outras partes interessadas. Os serviços devem ser diferenciados em:

- Serviços de negócio (fluxo de valor primário);
- Serviços de TI (fluxo de valor secundário).

Dado que o objetivo do Lean-IT é a redução de desperdício, os Serviços de TI são secundários (serviços de suporte) ao Serviços de negócio. Ou seja, se um serviço de TI não entrega valor ao negócio, significa ser um tipo de desperdício.

Time de sustentação

Gerente Técnico de Sustentação: gestor técnico responsável pela análise e atendimento das demandas de sustentação (incidentes e requisições), bem como da garantia dos níveis mínimos de serviço exigido para a aplicação.

Time de Sustentação: consiste em profissionais que realizam o trabalho de manutenção da disponibilidade, estabilidade e desempenho dos sistemas, tais como manutenções corretivas, perfectivas, adaptativas, integrativas e de interface, além de investigação de incidentes, resolução de problemas e apoio aos usuários, bem como atividades de suporte à manipulação de dados, monitoramento e operação das aplicações.

Artefatos

Backlog de Sustentação: é uma lista de todas as demandas (incidentes e requisições) reportada pelos usuários das aplicações de *software*. O Gerente Técnico de Sustentação é responsável pela análise e pelo atendimento das demandas, bem como da garantia dos níveis mínimos de serviço exigido para a aplicação.

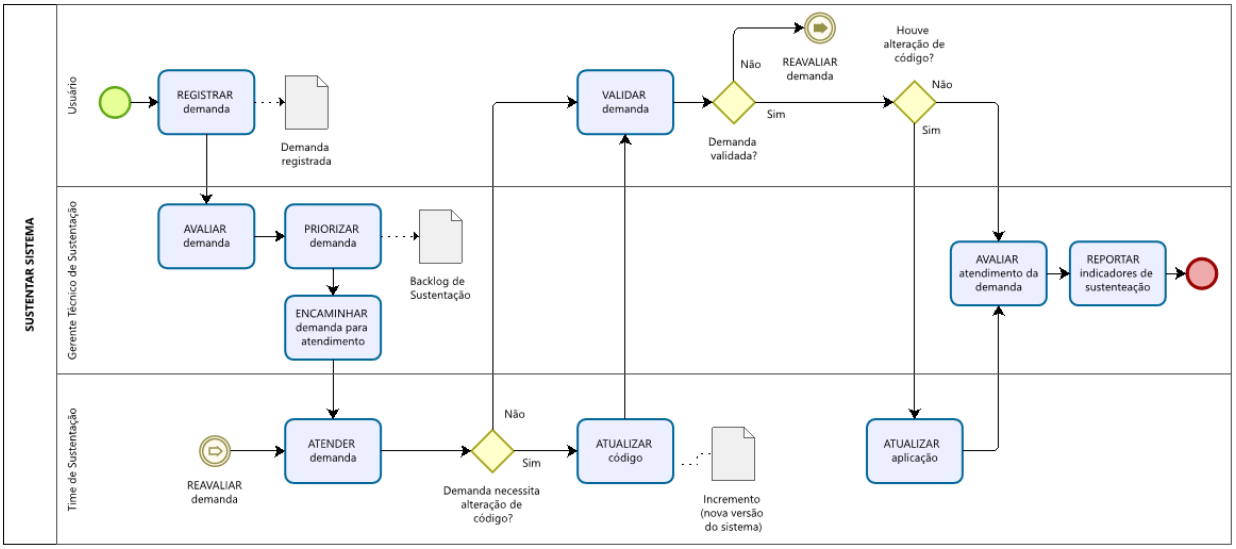
Incidentes: erro, falha ou indisponibilidade com parada total ou parcial da aplicação de *software*, de seus componentes ou de seus ambientes.

Requisições: solicitações de atendimento aos usuários das aplicações de *software*.

Incremento: é a correção de um ou vários incidentes do Backlog de Sustentação, em que foi necessária alteração de código da aplicação.

Artefato		**Incidente**			**Requisição**		
		Obrigatório	Opcional	Não se aplica	Obrigatório	Opcional	Não se aplica
Backlog de Sustentação		X	-	-	X	-	-
Incremento	Código-fonte	X	-	-	-	-	X
	Script de banco de dados	X	-	-	-	-	X
	Documentação técnica	X	-	-	X	-	-

Processo de Sustentação



10. Gestão de Mudança

A gestão de mudança é um processo que visa garantir que os métodos e procedimentos sejam utilizados de maneira eficiente, minimizando impactos ao negócio da organização advindos de mudanças nos serviços de TI que ocorram sem planejamento, priorização e revisão adequadas.

Considerando a abordagem desta MDS-Ebserh, que tem em seu cerne conceitos dos métodos ágeis, a gestão de mudança ocorre naturalmente nas etapas do processo de desenvolvimento e sustentação. Dessa forma as mudanças na aplicação são:

- Registradas no *Backlog do Produto* ou no *Backlog de Sustentação*;
- Aprovadas e priorizadas pelo *Product Owner* ou Gerente Técnico de Sustentação;
- Analisadas quanto ao impacto no planejamento e na operação da solução;
- Implementadas e testadas conforme o processo já definido; e
- Homologadas pelas áreas técnicas e negociais responsáveis.

A MDS-Ebserh permite a adequação do escopo e implementação das mudanças de forma ágil, e nesse sentido, considera a mudança como elemento natural ao processo de amadurecimento dos envolvidos com o projeto.

11. Processo de Implantação

Para a implantação de aplicações de *software* nas atividades, será utilizada uma metodologia específica para o ambiente da Ebserh, com fases e etapas pré-determinadas que otimizam e facilitam o acompanhamento do processo.

Etapas

Apresentação: reunião inicial para a apresentação dos envolvidos no processo e da metodologia, bem como para o alinhamento de expectativas.

Diagnóstico: consiste em uma série de reuniões que visam entender o ambiente em que a solução será implantada, além de identificar sistemas legados, necessidade de integrações e capacidade da infraestrutura de TI, quando for o caso.

Treinamento: capacitação na operação e utilização da aplicação de *software*.

Parametrização: atividades a serem executadas para definição de parâmetros e configuração do sistema a ser implantado.

Validação: aprovação da parametrização do sistema para liberação de uso.

Simulação: ensaio que testa a utilização do sistema realizadas em ambientes controlados (implantação), com o objetivo de encontrar problemas e/ou inconsistências na aplicação de *software*.

Preparação: reunião para organizar o início da utilização do sistema em ambiente produtivo.

Transição: início da utilização do sistema.

Acompanhamento: reuniões de monitoramento após o início da utilização do sistema.

Time de Implantação

Time de Implantação da Administração Central: equipe multidisciplinar da Administração Central, responsável pelo acompanhamento do projeto de implantação da aplicação de *software*, visando assegurar que as atividades previstas sejam executadas no prazo especificado.

Área de Negócio Local: equipe multidisciplinar da unidade onde a aplicação de *software* será implantada, incluindo a alta administração da unidade (diretores, coordenadores, superintendentes, gerentes etc.) ou responsáveis por ela designados.

Time de Implantação Local: equipe multidisciplinar da unidade onde a aplicação de *software* será implantada, responsável por configurar e parametrizar o sistema conforme as especificidades da unidade, bem como capacitar os demais colaboradores na utilização da solução.

Artefatos

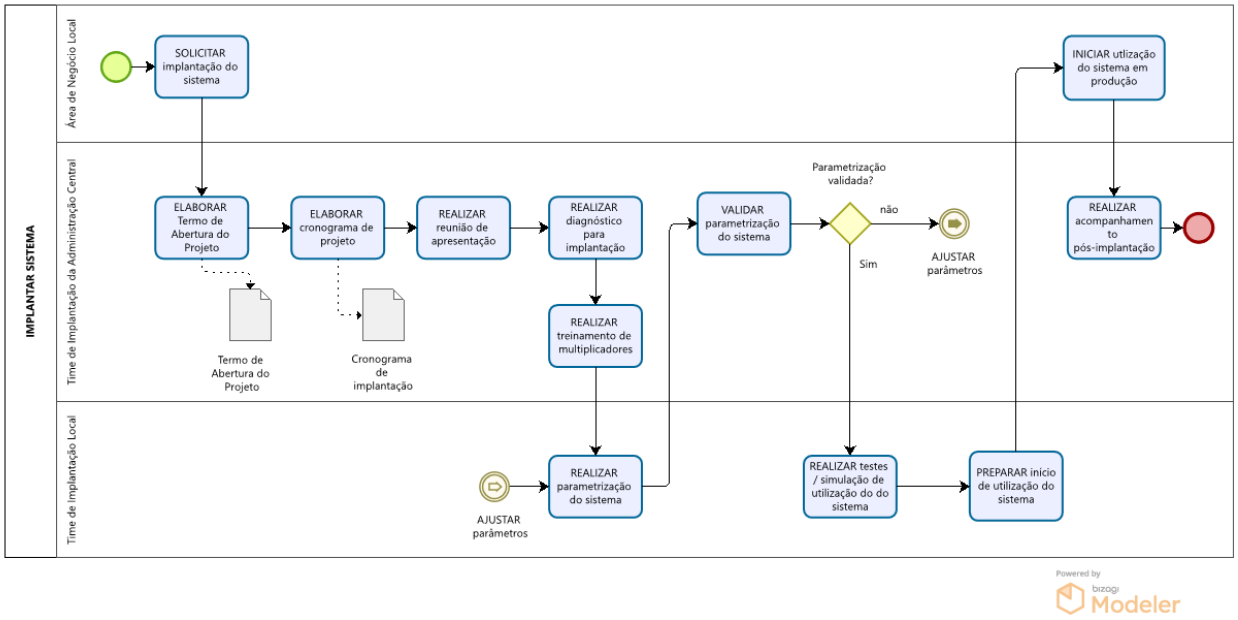
Termo de Abertura do Projeto (TAP): documento de autorização formal para início do projeto, que indica os objetivos, as áreas e pessoas envolvidas, as atividades e os prazos estimados para sua execução.

Cronograma de implantação: documento que descreve todas as atividades a serem cumpridas dentro do planejamento do projeto, incluindo prazos e responsáveis.

Relatório de acompanhamento: documento de acompanhamento da execução das atividades do projeto.

Artefato	Novo sistema			Novo módulo de sistema existente		
	Obrigatório	Opcional	Não se aplica	Obrigatório	Opcional	Não se aplica
Termo de Abertura do Projeto	X	-	-	-	X	-
Cronograma de implantação	X	-	-	X	-	-
Relatório de acompanhamento	X	-	-	X	-	-

Processo de Implantação



12. Papéis e Responsabilidades

A MDS-Ebserh contempla diversos papéis que executam, cada um, atividades e responsabilidades específicas dentro de cada processo dessa Metodologia. Para facilitar a compreensão do documento, será apresentada a seguir a relação das responsabilidades atribuídas a cada um dos papéis envolvidos. Os papéis apresentados representam uma adaptação dos *frameworks* à realidade operacional da Ebserh.

Ator	Papel	Responsabilidade
Área Requisitante	Product Owner	• Expressar claramente os itens do Backlog do Produto • Ordenar os itens do Backlog do Produto para alcançar melhor as metas e missões • Garantir o valor do trabalho realizado pelo Dev Team • Garantir que o Backlog do Produto seja visível, transparente, claro para todos, e mostrar o que o Time Scrum vai trabalhar a seguir • Garantir que o Dev Team entenda os itens do Backlog do Produto no nível necessário • Avaliar as entregas realizadas e os indicadores de desenvolvimento, inclusive as equipes de desenvolvimento, quando aplicável
Equipe de Desenvolvimento de Sistemas	*Scrum* Master	• Garantir que o Scrum seja entendido e aplicado
	Time de Desenvolvimento	• Realizar o trabalho de entregar uma versão usável que potencialmente incrementa o produto "Pronto" ao final de cada Sprint
Equipe de Sustentação de Sistemas	Gerente Técnico de Sustentação	• Garantir a análise e atendimento das demandas de sustentação (incidentes e requisições), bem como os níveis mínimos de serviço exigido para a aplicação
	Time de Sustentação	• Realizar o trabalho de manutenção da disponibilidade, estabilidade e desempenho dos sistemas, tais como manutenções corretivas, perfectivas, adaptativas, integrativas e de interface, além de investigação de incidentes, resolução de problemas e apoio aos usuários, bem como atividades de suporte à manipulação de dados, monitoramento e operação das aplicações
Equipe de Arquitetura de Sistemas	Arquitetos e Engenheiros de *Software*	• Definir, manter, implementar e configurar, em parceria com a equipe de infraestrutura, os padrões, o modelo de arquitetura, as rotinas e esteiras de integração contínua (CI) e entrega contínua (CD), sendo o apoio arquitetural da Ebserh para as Equipes de Desenvolvimento de Sistemas
Equipe de Implantação de Sistemas	Time de Implantação da Administração Central	• Realizar o acompanhamento do projeto de implantação • Assegurar que as atividades previstas sejam executadas no prazo especificado
	Área de Negócio Local	• Realizar o acompanhamento do projeto de implantação nas áreas de negócio da unidade (Administração Central ou Hospital Universitário)
	Time de Implantação Local	• Configurar e parametrizar o sistema, de acordo com as especificidades da unidade • Capacitar demais colaboradores na unidade para implantação do sistema

13. Ferramentas

Gestão de demandas

As demandas de desenvolvimento e sustentação de sistemas deverão ser registradas e acompanhadas por meio das ferramentas definidas pela Coordenadoria de Sistemas da Informação, de acordo com a definição de cada projeto.

As demandas de implantação de sistemas deverão ser registradas e acompanhadas por meio do Aplicativo de Gestão de Implantação de Sistemas - Agis (http://agis.ebserh).

A qualquer tempo, seja em função da revisão dos processos institucionais ou da evolução do parque tecnológico da Ebserh, as ferramentas poderão ser revistas e redefinidas, de acordo com as necessidades da DTI.

Controle de versão

O código fonte das aplicações de *software* deverão ser versionados em sistema definido pela Coordenadoria de Sistemas da Informação, estruturados nos repositórios, de acordo com a definição de cada projeto.

14. Versionamento

Repositórios

A organização dos repositórios da aplicação se dará da seguinte forma:

source: utilizado para aplicações monolíticas ou legadas onde a estrutura de front-end não pode ser separada do back-end.

back-end: utilizado para aplicações que rodam no servidor independentes de uma interface gráfica. Exemplo: API, CRON, Web Service etc.

front-end: utilizado para front-end web desacoplados do back-end, como aplicações ReactJS, Angular, VueJS etc.

mobile: utilizado para aplicações que irão rodar no dispositivo móvel.

docs: utilizado para o armazenamento de documentações do projeto.

NOTA: Se por algum motivo o projeto necessitar de mais de um repositório para qualquer dos tópicos listados acima, basta especializar o nome do repositório. Por exemplo: se dentro do projeto A tenho diversos módulos negociais em PHP e por questões técnicas o módulo de criptografia é em Java, basta neste caso criar os repositórios **back-end-php** e **back-end-java**.

Da mesma forma, se tenho um aplicativo nativo para iOS e outro para Android, basta criar os repositórios **mobile-ios** e **mobile-android**, assim como um **front-end-angular** e outro **front-end-react** caso necessário.

Os repositórios **source**, **back-end**, **front-end** e **mobile** são repositórios de código fonte. Em cada um deles deve ser criado um arquivo **README.md** contendo:

- Nome e descrição do projeto;
- Tecnologias utilizadas;
- Estrutura de pastas;
- Padrão de nomenclatura de arquivos;
- Passos para configuração do ambiente.

Branches

Cada repositório será composto por diversas *branches*, que são organizadas em duas categorias: **principal** e **auxiliar**.

Categoria principal

É composta pelas branches **dev, rc** e **main**. As branches da categoria principal devem ser protegidas contra *commits* e só poderão ser atualizadas através de **Pull Request**, mediante aceitação de aprovadores. Essas branches representam os ambientes de Desenvolvimento, Homologação e Produção, respectivamente.

dev: branch onde se concentram as novas funcionalidades/melhorias da próxima entrega. Após todas as features de uma demanda serem integradas na dev, deve ser feito a entrega em rc através de um Pull Request.

rc: branch pronta para ser homologada. A branch rc é atualizada com a dev quando o desenvolvimento de todas as atividades de uma demanda estiver concluído.

main**:** branch com código fonte estável/versão de produção. Após a branch rc ser homologada, deve ser aberto um Pull Request para atualização da main.

Categoria auxiliar

É composta basicamente pelas branches iniciadas em feature/*, hotfix/* e bugfix/*.

feature/*: feature branches são criadas a partir da dev. Esta categoria de branch é utilizada para o desenvolvimento de novas funcionalidades/melhorias. Quando as atividades da branch estiverem concluídas, ela deve ser integrada de volta à dev. Se por qualquer motivo a dev for atualizada com a main ou com a rc, todas as demais feature em desenvolvimento deverão ser atualizadas com a dev.

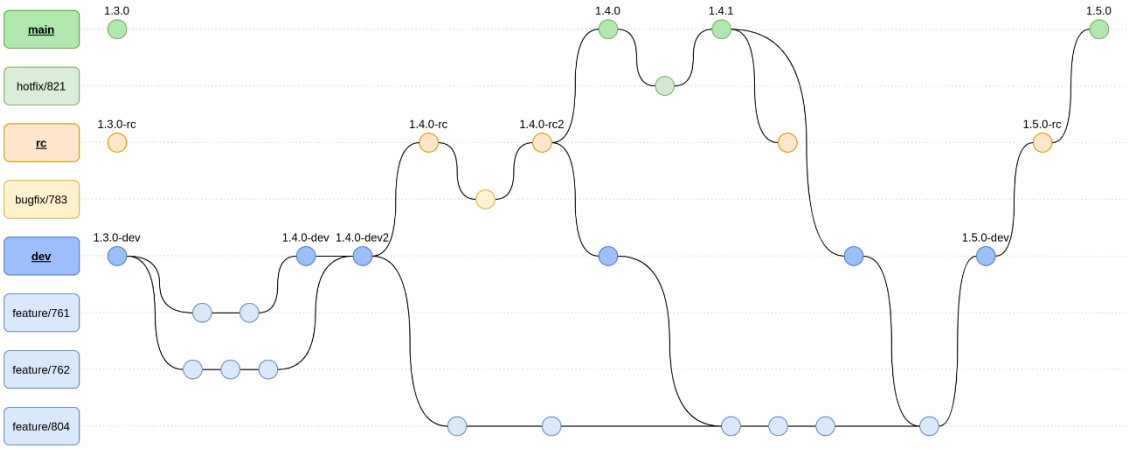
bugfix/*: bugfix branches são criadas a partir da branch rc. Esta categoria de branch é utilizada para correções de bugs encontrados na etapa de homologação. Após correção dos bugs, ela deve ser integrada de volta à rc. Quando a rc for atualizada devido a algum bugfix, a branch dev deverá ser atualizada com a rc.

hotfix/*: hotfix branches são criadas a partir da branch main. Esta categoria de branch é utilizada para correções de erros críticos encontrados em produção. Quando a main for atualizada devido a um hotfix, as branches rc e dev deverão ser atualizadas com a main.

NOTA**:** O nome de uma nova branch auxiliar deverá conter um dos três prefixos acima, mais o número da demanda de solicitação de nova funcionalidade, melhoria ou correção de bug, que deu origem à branch. Exemplo: feature/761.

Os commits destas branches devem ser enviados para a branch remota com frequência, a fim de diminuir a incidência de conflitos de merge e facilitar o acompanhamento do trabalho que está sendo realizado.

Quando o trabalho em uma branch auxiliar estiver concluído, deverá ser aberto um Pull Request para a branch principal que deu origem a ela. Após passar pelos fluxos de teste e validação, um aprovador deverá aceitar ou rejeitar o Pull Request. Sempre que um Pull Request for aprovado, uma nova tag deve ser criada seguindo o padrão de tags da Ebserh e a branch deverá ser excluída.



1. Uma branch auxiliar (feature, bugfix, hotfix) sempre é criada a partir de uma branch principal (dev, rc, main).
2. Quando uma feature é concluída, ela deve ser mesclada de volta na branch principal dev.
3. Antes de atualizar a branch dev com uma feature, deve ser criada uma nova tag -dev na branch de origem.

4. Quando um conjunto de features estiver pronto e integrado à dev, a rc deverá ser atualizada com a dev para que as features sejam homologadas.
5. Sempre que a rc for atualizada com a dev, deve ser criada uma nova tag -rc.
6. Caso um problema seja detectado na etapa de homologação, uma branch bugfix deverá ser criada a partir da rc e mesclada de volta após a correção.
7. Sempre que a rc for atualizada com um bugfix, deve ser criada uma nova tag -rc e a dev e features em desenvolvimentos devem ser atualizadas.
8. Após homologação da rc, esta deve ser mesclada à main para disponibilização em produção.
9. Caso um problema seja detectado em produção, uma branch de hotfix deve ser criada para correção diretamente a partir da main.
10. Após a conclusão do hotfix, ele é mesclado de volta para a main e as demais branches principais e auxiliares em desenvolvimento devem ser atualizadas.
11. Sempre que a main for atualizada, deve ser criada uma nova tag estável.

Commit

Uma mensagem de commit consiste na descrição das mudanças realizadas em função de uma parte do trabalho. Serve também como uma forma dos membros do time comunicarem o que está sendo feito. Uma mensagem de commit pode ser adicionada através do comando:

git commit -m "<assunto>"

Por padrão, as mensagens de commit da Ebserh são divididas em cinco partes:

Número da demanda: número da demanda que deu origem ao commit.

Tipo da modificação:

- FEATURE: adição de novas funcionalidades.
- BUGFIX ou HOTFIX: correção de bugs.
- REFACTOR: refatoração de código.

Ícone representativo do tipo da modificação [opcional]: conforme guia do GitMoji disponível em https://gitmoji.dev.

- ✨ (:sparkles:) - Criação de funcionalidades.
- 🐛 (:bug:) - Correção de bugs.
- ♻️ (:recycle:) - Refatoração de código.

Breve descrição das mudanças realizadas: a mensagem deve ser escrita no tempo verbal presente do indicativo. Deve indicar o que o commit faz, e não o que foi feito ou como foi feito. Exemplo:

✓ Altera a validação de datas no formulário de inscrição

✗ Alteração na validação de datas no formulário de inscrição

✗ Alterada validação de datas no formulário de inscrição

✗ Alterado o 'if (dtNascimento <= now())' para "if (dtNascimento <= dtCadastro)'

Detalhamento das mudanças realizadas [opcional]: caso seja necessário, poderá ser adicionada à mensagem maiores informações do que foi realizado. Neste caso, deixe uma linha em branco para separar o assunto do corpo do commit. Isso pode ser feito de diferentes formas. Exemplo:

 # Em um comando com apenas um parâmetro

git commit -m "<assunto>

<detalhamento>"

 # Em um comando com dois parâmetros

git commit -m "<assunto>" -m "<detalhamento>"

 # Em um comando sem parâmetros

git commit

Neste último caso, o editor configurado no Git será aberto para que entre com o assunto e detalhamento do commit.

Para escrever uma mensagem de commit clara e informativa:

1. Separe o assunto do corpo com uma linha em branco.
2. Tente limitar a linha de assunto a 50 caracteres.
3. Use o tempo verbal presente do indicativo na linha de assunto.

4. Envolva o corpo em blocos de 72 caracteres.

5. Use o corpo para explicar o quê e o porquê ao invés de como.

Formato do assunto das mensagens de commit:

#[DEMANDA][TIPO] :icone: <Mensagem>

Exemplo:

git commit -m "#671 [FEATURE] :sparkles: Adiciona pesquisa ao cadastro de hospitais"

Tags de versionamento

Em gerenciamento de código fonte com Git, uma tag nada mais é do que um ponteiro ou um apelido para um commit específico. Estes apelidos podem ser utilizados para sinalizar marcos ao longo do desenvolvimento do projeto. Uma versão estável ou em desenvolvimento do *software* são alguns dos marcos possíveis.

Versão estável

A definição das tags de versão do projeto deverá seguir o Versionamento Semântico 2.0.0:

Dado um número de versão MAJOR.MINOR.PATCH, incremente a:

versão Maior (MAJOR): quando fizer mudanças incompatíveis na API, versão Menor (MINOR): quando adicionar funcionalidades mantendo compatibilidade, e versão de Correção (PATCH): quando corrigir falhas mantendo compatibilidade. Rótulos adicionais para pré-lançamento(pre-release) e metadados de construção(build) estão disponíveis como extensão ao formato MAJOR.MINOR.PATCH.

Em outras palavras

Um número de versão normal DEVE ter o formato de X.Y.Z, onde X, Y, e Z são inteiros não negativos, e NÃO DEVE conter zeros à esquerda. X é a versão Maior, Y é a versão Menor, e Z é a versão de Correção. Cada elemento DEVE aumentar numericamente. Por exemplo: 1.9.0 -> 1.10.0 -> 1.11.0.

2 . 6 . 1

| | |

| | └——> PATCH

| | Incrementado a cada nova versão de correção de bugs.

| |

| └————> MINOR

| Incrementado sempre que é feito uma mudança que não quebre a

| compatibilidade com versões anteriores.

└————————> MAJOR

Incrementado sempre que uma grande mudança ocorre como uma

mudança de Framework ou se a mudança causar a quebra de

compatibilidade com versões anteriores.

Sempre que a branch main for atualizada, deve ser criada uma nova tag estável. Exemplo: 2.6.0

Sempre que um Pull Request de uma branch hotfix/* for aceito, uma nova tag incrementando o PATCH VERSION deverá ser criada. Exemplo: 2.6.1

Quando um Pull Request de uma branch feature/* for aceito, uma nova tag incrementando o MINOR VERSION deverá ser criada. Sempre que o MINOR VERSION for incrementado, o PATCH VERSION deverá ser zerado. Exemplo: 2.7.0

Versão pré-lançamento

Uma versão de Pré-Lançamento (pre-release) PODE ser identificada adicionando um hífen e uma série de identificadores separados por ponto imediatamente após a versão de Correção. O identificador DEVE incluir apenas caracteres alfanuméricos e hífen [0-9A-Za-z-] e NÃO DEVE ser vazio. O indicador numérico NÃO DEVE incluir zeros à esquerda. Uma versão de Pré-Lançamento tem precedência inferior à versão normal a que está associada. Uma versão de Pré-Lançamento indica que a versão é instável e pode não satisfazer os requisitos de compatibilidade pretendidos, como indicado por sua versão normal associada. Exemplo: 2.8.0-rc, 2.8.0-dev.1.

2 . 8 . 0 -dev.1

└—|—|┘

└--|-——> PRÉ-LANÇAMENTO

| Indica uma versão ainda em desenvolvimento.

| Pode estar pendente de testes e até mesmo de implementações.

|

└——> O número ao final é incrementado para cada nova entrega da mesma

demanda/funcionalidade.

Antes de atualizar a branch dev com uma feature, deve ser criada uma nova tag de pré-lançamento -dev na branch de origem. Exemplo: 2.8.0-dev.

Sempre que a branch rc for atualizada com a dev ou com um bugfix, deve ser criada uma nova tag de pré-lançamento -rc. Exemplo: 2.8.0-rc.

Uma tag de pré-lançamento é criada incrementando a última versão estável de acordo com as regras do versionamento semântico e adicionando o identificador -rc ou -dev. Por exemplo, se a última tag estável é a versão 2.7.0, ao trabalhar em uma nova funcionalidade, enquanto a nova versão não fica pronta e estável, a versão a ser utilizada será 2.8.0-dev.

15. Normativos

O processo de desenvolvimento, sustentação e implantação de aplicações de *software* no âmbito da Ebserh deverá ser balizado pela legislação e pelos atos normativos vigentes, bem como pelas boas práticas no desenvolvimento de sistemas recomendadas para a Administração Pública, tais como:

- Lei nº 13.709, de 14 de agosto de 2018;
- Instrução Normativa SGD/ME nº 94, de 23 de dezembro de 2022;
- Instrução Normativa nº 1, de 27 de maio de 2020;
- Política de Segurança da Informação e Comunicações da Ebserh (PSI-Ebserh);
- Padrões de Interoperabilidade de Governo Eletrônico (ePing);
- Modelo de Acessibilidade em Governo Eletrônico (eMag);
- Padrões Web em Governo Eletrônico (ePwg);
- Guia de Segurança em Aplicações Web para LGPD;
- Guia de Requisitos Mínimos de Segurança e Privacidade para Aplicativos Móveis para LGPD;
- Guia de Requisitos e deObrigações quanto a Segurançada Informação e Privacidade para LGPD;
- Guia do Framework de Privacidade e Segurança da Informação para LGPD;
- Guia de Requisitos Mínimos de Segurança e Privacidade para APIs para LGPD.