UNIVERSIDADE FEDERAL DE SERGIPE
DEPARTAMENTO DE COMPUTAÇÃO-DCOMP
Engenharia de Software II

Docente: Dr. Glauco de Figueiredo Carneiro
Discentes:
Álex Santos Alencar - 202300061518
Ellen Karolliny dos Santos - 202300114326
Gabriel Luiz Santos Gama Barreto - 202300114335
Gabriel Ramos de Carvalho - 202300061920
João Andryel Santos Menezes - 202300061652
Larissa Batista dos Santos - 202300061705
Paloma dos Santos - 202300061723
Rauany Ingrid Santos de Jesus - 202300061760

Atividade 3 – Parte 1

Tabela 1 - Contribuição de cada membro da equipe.

Nome	**Matrícula**	**Descrição da atividade**
Álex Santos Alencar	202300061518	Diagnóstico e análise do fluxo atual de desenvolvimento.
Ellen Karolliny dos Santos	202300114326	Definição de automação e seus tipos.
Gabriel Luiz Santos Gama Barreto	202300114335	Diagnóstico e análise de automação no repositório do *DeepResearch.*
Gabriel Ramos de Carvalho	202300061920	Diagnóstico e análise dos riscos de regressão do projeto.
João Andryel Santos Menezes	202300061652	Diagnóstico e análise de automação no repositório.
Larissa Batista dos Santos	202300061705	Diagnóstico e análise de automação no repositório.
Paloma dos Santos	202300061723	Diagnóstico e análise dos gargalos manuais e limitações operacionais.
Rauany Ingrid Santos de Jesus	202300061760	A importância e as vantagens da automação e organização do documento de entrega.

Fonte: Elaboração própria (2026)

1- O que é Automação?

A automação é a aplicação de tecnologias e processos para converter atividades manuais e repetitivas em fluxos executados por sistemas ou softwares, operando com mínima intervenção humana. O objetivo central é garantir que as tarefas sejam realizadas de forma padronizada e eficiente, seguindo rigorosamente regras predefinidas. Na prática, o funcionamento baseia-se em uma lógica condicional onde, sempre que um evento ou condição específica acontece, uma ação correspondente é disparada automaticamente.

No desenvolvimento de software, a automação está associada à execução automática de etapas como: Compilação de código, Execução de testes,Verificação de padrões de código (lint), Geração de builds, Publicação ou deploy.

Estruturalmente, esse mecanismo depende de quatro elementos: uma entrada ou gatilho (como um commit no GitHub), regras que definem o comportamento do sistema, o processo automatizado propriamente dito e a saída, que representa o resultado final, como um relatório ou um build concluído. Dependendo da complexidade, a automação pode variar de scripts simples para tarefas isoladas até fluxos de processos encadeados (build, teste e deploy), podendo ser acionada por eventos específicos ou funcionar de forma contínua para monitorar e reagir a qualquer alteração no ambiente.

2 - A Importância Estratégica da Automação

A automação no ciclo de engenharia de software deixou de ser um diferencial competitivo para se tornar uma necessidade estratégica. Sua importância se manifesta em múltiplas dimensões que impactam diretamente a qualidade, velocidade e confiabilidade do software entregue.

Garantia de Qualidade Constante - Cada vez que um desenvolvedor envia uma alteração de código, a automação atua como um sistema imunológico digital. Testes automatizados são executados imediatamente, validando não apenas a nova funcionalidade, mas verificando se nenhuma função existente foi quebrada. Isso cria uma barreira de proteção contra o retrabalho que, em processos manuais, só seriam descobertas tardiamente.

Aceleração do Ciclo de Desenvolvimento - Processos que custariam horas de trabalho manual - como build, execução de testes, análise de código e deploy - são reduzidos a minutos. Essa eficiência permite que as equipes realizem dezenas de integrações por dia, facilitando práticas ágeis e entregas incrementais. O tempo poupado é reinvestido em inovação, não em tarefas repetitivas.

Padronização e Eliminação de Erros Humanos - Ao definir pipelines de CI/CD, criamos um ambiente consistente e reprodutível para build, teste e implantação. Isso elimina variáveis ambientais e diferenças entre configurações de desenvolvedores, garantindo que o comportamento do software seja previsível em todos os estágios.

Feedback Imediato e Cultura de Melhoria Contínua - Quando um pipeline automatizado falha, o desenvolvedor recebe notificação em tempo real, podendo corrigir o problema enquanto o contexto ainda está fresco na memória. Esse ciclo rápido de feedback estimula a aprendizagem

contínua e a responsabilidade coletiva pela qualidade do código, formando uma cultura de excelência técnica.

Redução de Riscos Operacionais - Releases manuais são eventos de alto estresse e alto risco. A automação transforma o deploy em um processo rotineiro, controlado e reversível. Mudanças menores e mais frequentes significam que, se algo der errado, o escopo do problema é limitado e o rollback é simplificado.

A automação em CI/CD não é apenas sobre tecnologia - é sobre mudança cultural. Ela transforma o desenvolvimento de software de uma arte manual imprevisível em uma disciplina de engenharia confiável.

3 - Investigação e Evidências

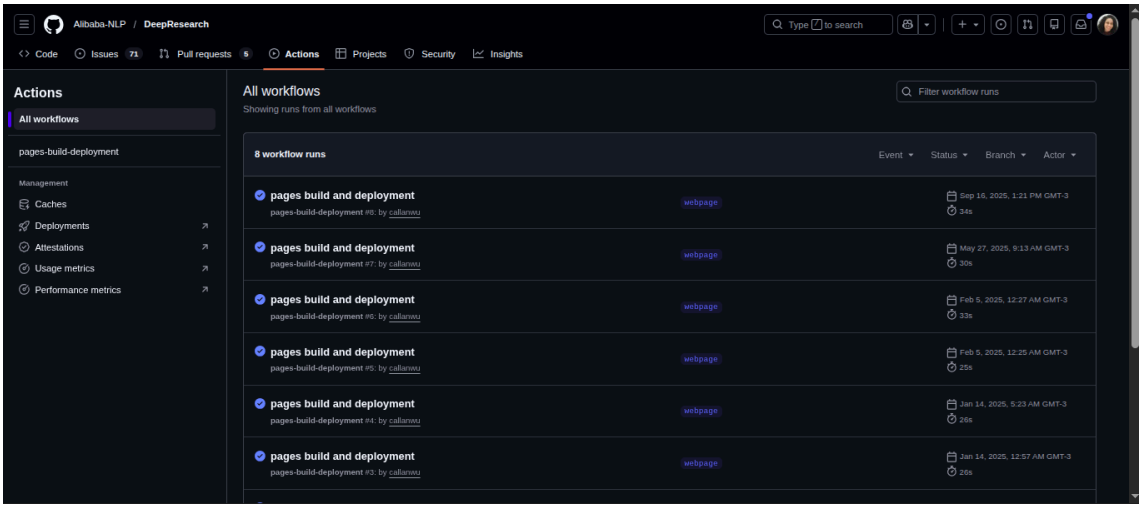
3.1. Diagnóstico Geral sobre a Automação

Ao analisar o repositório do *Deep Research*, foi observado que não existe uma verificação automática para encontrar erros no código. A pasta que normalmente guarda essas instruções *(.github/workflows* no *Github Actions)* não foi encontrada e nenhum arquivo *.yml* responsável por processos de Integração Contínua (CI) responsável por testes, lints e validações, nem de Implantação Contínua (CD) como *deploys* e correções automáticas no código fonte. Isso significa que, quando um programador faz uma alteração, não há um "robô" que testa as modificações sozinho para dizer se está certo ou errado. A garantia de que o código funciona vem dos testes manuais feitos pelos próprios desenvolvedores.

3.2. Atualização Automática do Site

Apesar da ausência de testes para o código principal, foi identificado no github actions um fluxo de Implantação Contínua (CD) chamado *pages-build-deployment*. A função dele é exclusiva para a documentação. Sempre que os manuais ou textos explicativos são alterados, o sistema atualiza sozinho o site oficial do projeto. O benefício direto é manter as instruções de uso sempre em dia para o público, sem que a equipe precise fazer essa atualização manualmente.

Figura 1 - Página do GitHub Actions



Fonte: Elaboração própria (2026)

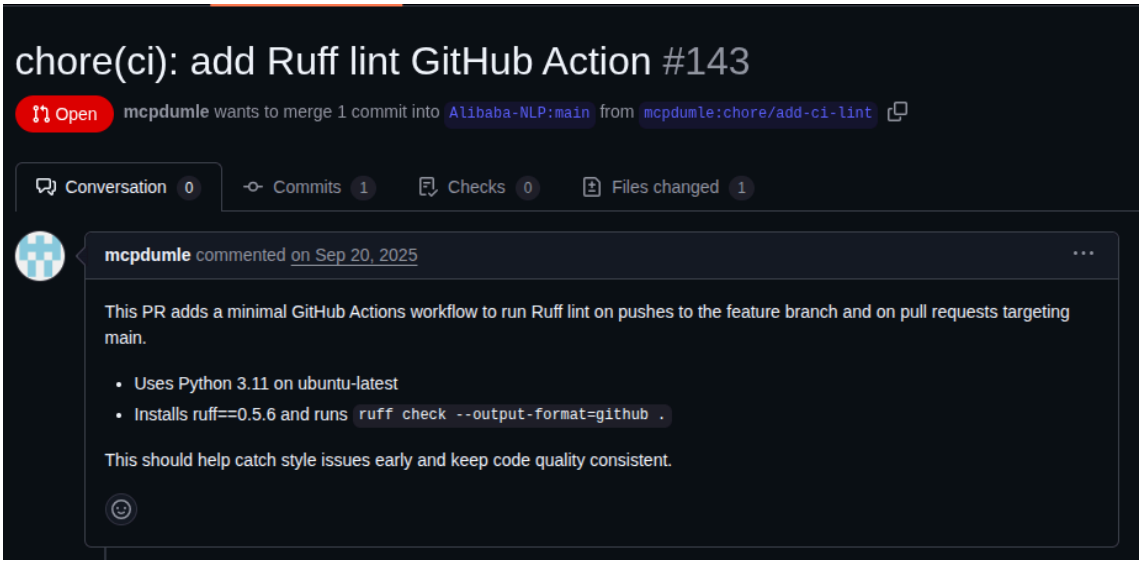
3.3. Automação do Ambiente Local

Mesmo sem automação configurada no repositório, detectou-se uma ferramenta para organizar o ambiente local: o arquivo *run_react_infer.sh*. Ele foi identificado como a peça principal de engenharia do projeto. Esse script substitui as configurações manuais difíceis, pois prepara o ambiente, baixa o que é necessário e coloca a inteligência artificial para rodar. Basicamente, ele serve para garantir que o programa funcione da mesma maneira no computador de qualquer pessoa que baixar o projeto.

3.4. Análise dos Pull Requests

A análise do histórico de pull requests confirmou o pensamento inicial. Não foram encontradas validações automáticas de testes nas aprovações de código. No entanto dentre as solicitações pendentes se destacou o *Pull Request #143*, que traz uma proposta para adicionar uma ferramenta de verificação automática (*Linter*). A existência desse pedido externo corrobora o fato de que o projeto não possui sistemas nativos para filtrar erros de código. Essa ferramenta atuaria como um filtro de qualidade técnica, mas, como ainda não foi implementada, o processo continua sem automação.

Figura 2 - Pull Request #143



Fonte: Elaboração própria (2026)

4 - Fluxo, Riscos e Gargalos

4.1 - Fluxo de Desenvolvimento

Atualmente, o fluxo é focado na publicação de resultados e execução local, seguindo estes passos:

Codificação: O desenvolvedor realiza alterações nos scripts de pesquisa ou na lógica do agente (deep_research/).

Validação de Ambiente: O desenvolvedor utiliza o script run_react_infer.sh para configurar o ambiente e validar se o modelo de inferência está rodando corretamente na máquina local.

Submissão (Pull Request): O código é enviado para o GitHub.

Integração de Documentação: Se houver alteração na documentação, o workflow pages-build-deployment é acionado automaticamente para atualizar o site do projeto.

Merge: O código é integrado à branch principal sem passar por baterias de testes automatizados no servidor.

4.2. Riscos de Regressão

Um dos riscos identificados é a Regressão de Desempenho do Modelo (Model Drift), onde o agente não necessariamente trava, mas se torna 'menos inteligente'. Pequenas refatorações no código podem alterar a forma como o contexto é processado ou truncado, levando a alucinações e falhas na resolução de problemas.

Por fim, a Regressão Funcional de Ferramentas expõe que a comunicação, realizada via troca de strings ou JSON, carece de contratos de interface fortes; consequentemente, se uma ferramenta externa alterar o formato de sua resposta, o agente perderá a capacidade de processar a informação corretamente.

4.3. Gargalos Manuais e Limitações Operacionais

A inovação algorítmica do projeto está à frente de sua maturidade operacional, gerando diversas barreiras manuais.

Dependência de Code Review (Gargalo Humano): A validação da lógica e da segurança recai quase totalmente sobre a revisão humana, criando um gargalo onde PRs se acumulam aguardando a atenção limitada dos especialistas.

Configuração de Ambiente (Onboarding): O processo de setup é manual (Conda) e não padronizado via Docker, levando a problemas recorrentes de "funciona na minha máquina" e dificultando a entrada de novos desenvolvedores.

Gestão de Credenciais e Testes: O projeto não roda "offline" e depende de chaves de API (caras e sensíveis). Como o GitHub Actions não pode acessar segredos de forks externos, contribuidores não conseguem rodar testes de integração no CI, postergando a validação real para o mantenedor.

Benchmarking Manual: A avaliação da qualidade do modelo (se ele ficou mais ou menos preciso) é feita através da execução manual de scripts locais, desconectada do fluxo de CI. Isso faz com que decisões de merge sejam tomadas "às cegas" em relação ao impacto na qualidade cognitiva.

Limitação de Hardware: A exigência de GPUs de classe servidor para rodar o modelo localmente impede que desenvolvedores individuais contribuam com mudanças profundas na lógica, restringindo a comunidade a correções superficiais.

5. Materiais Complementares

Para visualizar todos os artefatos utilizados na análise, disponibilizou-se um repositório que reúne os arquivos de apoio, registros gerados e análises detalhadas. O acesso aos seguintes materiais pode ser realizado pelo endereço: https://github.com/GabrielGamaUFS/Engenharia_Software_2025-2_DeepResearch_atividade3_parte1

O vídeo a seguir apresenta a explicação da análise da automação do *DeepResearch*: https://drive.google.com/file/d/16-FSiWnY2sPF4_BGmUp_7IoLqNMQBNzv/view?usp=drivesdk

REFERÊNCIAS

SERVICENOW. What is Automation? Disponível em: https://www.servicenow.com/br/platform/what-is-automation.html. Acesso em: 30 jan. 2026.

DORA.Accelerate State of DevOps Report. Google Cloud. Disponível em: https://cloud.google.com/devops/state-of-devops. Acesso em: 30 jan 2026.

SONG, Jimmy. Deep Research. Disponível em: https://jimmysong.io/ai/deep-research/. Acesso em: 30 jan. 2026.

ALIBABA-NLP. DeepResearch. GitHub, 2025. Disponível em: https://github.com/Alibaba-NLP/DeepResearch. Acesso em: 30 jan. 2026.