Revista Eletrônica de Iniciação Científica em Computação, 2026, 24:1
ISSN: 3085-8461 • doi: 10.5753/reic.2026.7182

Artigo de Pesquisa/Research Paper

Metodologia Ágil e IA em Sistemas Multiagentes: Avaliação Experimental de uma Arquitetura para Planejamento e Execução de Sprints

Agile Methodology and AI in Multi-Agent Systems: Experimental Evaluation of an Architecture for Sprint Planning and Execution

Elysson Alves de Lacerda [Universidade Federal do Ceará - Campus Quixadá (UFC) | *elyssonalvs@alu.ufc.br*]
Gustavo Almeida Monteiro [Universidade Federal do Ceará - Campus Quixadá (UFC) |*gustavoam@alu.ufc.br*]
Franciel Silveira Penha de Vasconcelos [Universidade Federal do Ceará - Campus Quixadá (UFC) |*francielsilveira@alu.ufc.br*]
Marcos Antonio de Oliveira [Universidade Federal do Ceará - Campus Quixadá (UFC) |*marcos.oliveira@ufc.br*]

Universidade Federal do Ceará - Campus Quixadá (UFC) Av. José de Freitas Queiroz, 5003-Cedro, 63902-580-Quixadá-CE

Resumo. Este artigo apresenta um estudo experimental sobre uma arquitetura de Sistemas Multiagentes (SMAS) integrada à plataforma N8N, projetada para otimizar processos em metodologias ágeis, com foco no Scrum. A arquitetura aborda desafios críticos como planejamento de sprints e alocação de recursos, utilizando a técnica Bart para elicitação de requisitos e agentes baseados em IA para fluxos de trabalho personalizáveis. Esta versão estendida valida a proposta original, comparando três modelos de implementação (*TextClassifier*, *Sub-agentes* e *Webhook*) através de um estudo de simulação robusto em cenários de complexidade variável. Os resultados experimentais demonstram que o modelo Webhook oferece escalabilidade e adaptabilidade superiores, automatizando a priorização e atribuição de tarefas e reduzindo a carga cognitiva de *Scrum Masters* e *Product Owners*.

Abstract. This paper presents an experimental study of a Multi-Agent System (MAS) architecture integrated with the N8N platform to optimize administrative and managerial processes in agile methodologies, specifically Scrum. The architecture addresses challenges like sprint planning and resource allocation, using the Bart technique for requirements elicitation and AI-based agents for customizable workflows. This extended version validates the original proposal by comparing three implementation models (TextClassifier, Sub-agents, and Webhook) through a robust simulation study across varied complexity scenarios. Experimental results demonstrate that the Webhook model provides superior scalability and adaptability, automating task prioritization and assignment, thereby reducing the cognitive load on Scrum Masters and Product Owners.

Palavras-chave: Sistemas Multiagentes, Metodologia Ágil, Scrum, N8N, Planejamento de Sprint, IA

Keywords: Multi-Agent Systems, Agile Methodology, Scrum, N8N, Sprint Planning, AI

Received: 13 December 2025 • **Accepted:** 13 March 2026 • **Published:** 24 April 2026

1 Introdução

A adoção de metodologias ágeis, notavelmente o *Scrum*, tornou-se uma prática predominante, sendo considerada a principal abordagem ágil na indústria de *software* [Ali *et al*., 2017]. Contudo, a crescente complexidade de processos administrativos e gerenciais exige soluções inovadoras para otimização de recursos e redução de erros humanos [Laplante and Kassab, 2022].

O planejamento de *sprints*, uma cerimônia central do *Scrum*, exemplifica esse desafio. Esta atividade exige que *Scrum Masters* e *Product Owners* (POs) processem um grande volume de informações, incluindo a priorização de itens do *backlog*, a alocação de recursos da equipe e a identificação de dependências entre tarefas [Nawaz Aslam, 2023]. Conforme identificado no trabalho original [Lacerda *et al*., 2025], esse processo, muitas vezes manual, aumenta o tempo de planejamento e a sobrecarga de trabalho.

A literatura tem explorado a automação de tarefas do *Scrum* usando Inteligência Artificial (IA) e Sistemas Multiagentes (SMAs). Abordagens como a de [Gunga *et al*., 2013] propuseram arquiteturas de SMAs para o *Scrum*, enquanto [Lin *et al*., 2015] apresentaram sistemas inteligentes para distribuição de tarefas. No entanto, muitas soluções falham em prover flexibilidade e integração simplificada com as ferramentas modernas de *workflow*.

Para endereçar essa lacuna de flexibilidade, este trabalho propõe e avalia uma arquitetura que combina a autonomia dos SMAs com a plataforma de automação *N8N* [n8n.io, 2025]. A arquitetura permite a criação de um agente inteligente capaz de executar *workflows* de planejamento de *sprint*, desde a elicitação de requisitos (usando a técnica *Bart* [Gerstberger *et al*., 2024]) até a alocação final de tarefas.

Este artigo é uma versão estendida e substancialmente revisada do trabalho '*Agile Methodology and AI in Multi-Agent Systems: An agent for planning and executing sprints*', originalmente apresentado no 19th *Workshop-School on Agents, Environments, and Applications* (WESAAC 2025) [Lacerda *et al*., 2025].

Enquanto o trabalho original focou na concepção teórica

e na proposta das 3 arquiteturas, esta versão apresenta as seguintes contribuições inéditas:

- A implementação detalhada da terceira arquitetura, o escolhido, *Webhook*), de forma detalhada;
- O projeto e a execução de um estudo de simulação robusto, utilizando três cenários de complexidade variada para testar a eficácia e a escalabilidade da arquitetura;
- Uma análise experimental quantitativa comparando o desempenho da arquitetura em métricas de tempo de resposta e taxa de sucesso.

A seguir, este artigo está organizado da seguinte forma: A Seção 2 apresenta a fundamentação teórica. A Seção 3 detalha as três arquiteturas, bem como a escolha da melhor arquitetura. A Seção 4 descreve o desenho do estudo de simulação e os cenários de teste. A Seção 5 apresenta e discute os resultados experimentais. Por fim, a Seção 6 conclui o trabalho.

2 Fundamentação Teórica

2.1 Metodologia Ágil e Scrum

O *Scrum*, uma metodologia ágil, organiza o desenvolvimento de *software* pelos diversos ciclos iterativos chamados *sprints*. *Scrum Masters* e *Product Owners* enfrentam desafios como consolidação manual de relatórios e priorização de tarefas, podendo causar atrasos e erros [Wilmann and Sterling, 2005]].

A integração de IA e SMAs pode transformar a gestão de *sprints*, automatizando tarefas rotineiras, oferecendo *insights* baseados em dados e otimizando a alocação de recursos, reduzindo a sobrecarga cognitiva [Nawaz Aslam, 2023]. SMAs permitem monitoramento em tempo real de indicadores como velocidade e *burndown*, possibilitando ajustes dinâmicos às prioridades, alinhados ao princípio ágil de adaptação a mudanças [Nawaz Aslam, 2023].

2.2 Sistemas Multiagentes

Sistemas Multiagentes (SMAs) são uma área da inteligência artificial que estuda a interação de agentes autônomos para resolver problemas complexos [Gerstberger *et al*., 2024]. Em contextos administrativos, SMAs automatizam tarefas como alocação de recursos e monitoramento de métricas, reduzindo erros e aumentando a escalabilidade [Balaji and Srinivasan, 2010].

No *Scrum*, SMAs têm sido usados para otimizar processos ágeis. [Gunga *et al*., 2013] propuseram uma arquitetura de SMA para o *Scrum*, com agentes representando papéis como *Scrum Master* e *Product Owner*, automatizando planejamento de *sprints* e alocação de tarefas. [Lin *et al*., 2015] apresentaram um sistema inteligente que extrai metadados e distribui tarefas em equipes *Scrum*. Simulações baseadas em SMAs também modelam a produtividade, com agentes interagindo com *backlogs* para otimizar *sprints* [Lin *et al*., 2015].

Comparado aos trabalhos de [Gunga *et al*., 2013], que focam em automação básica sem integração externa, nossa proposta preenche lacunas ao integrar *N8N* para *workflows* visuais. Da mesma forma, [Lin *et al*., 2015] limitam-se à modelagem de produtividade sem monitoramento em tempo real ; nossa abordagem adiciona agentes para simulações de cenários, reduzindo gargalos em contextos ágeis voláteis.

2.3 Arquitetura de Software para Sistemas Multiagentes

Em Sistemas Multiagentes (SMAs), a arquitetura estabelece a forma como agentes autônomos são organizados, comunicam-se e coordenam suas ações dentro do sistema, constituindo um elemento determinante para garantir propriedades como eficiência, escalabilidade e adaptabilidade [Balaji and Srinivasan, 2010]. Nesse contexto, a definição arquitetural orienta não apenas a estrutura do sistema, mas também os mecanismos de interação e cooperação entre os agentes.

Abordagens arquiteturais tradicionais para SMAs frequentemente utilizam *frameworks* especializados, como o *Java Agent Development Framework* (JADE), que provê uma infraestrutura baseada em *Java* e protocolos padronizados de comunicação, como o *FIPA-ACL*, facilitando a troca de mensagens e a coordenação entre agentes [Jennings, 2000]. Entretanto, ambientes computacionais contemporâneos têm demandado arquiteturas mais flexíveis e integráveis com ecossistemas de software amplamente utilizados.

Nesse cenário, a *Service-Oriented Architecture* (SOA) apresenta-se como uma alternativa adequada para contextos heterogêneos, nos quais agentes podem atuar tanto como provedores quanto como consumidores de serviços, promovendo modularidade, reutilização e interoperabilidade entre componentes do sistema [Aziz, 2010]. Alinhado a essa perspectiva, o presente trabalho adota uma abordagem arquitetural baseada na orquestração de microsserviços utilizando a plataforma *N8N*, que possibilita a integração de diferentes ferramentas e serviços por meio de *APIs REST* e *webhooks*, os quais funcionam como mecanismos de comunicação entre os agentes.

A arquitetura proposta fundamenta-se no modelo deliberativo *BDI* (*Belief–Desire–Intention*), amplamente utilizado na modelagem de agentes inteligentes. No contexto da implementação com *N8N*, as *crenças* (*beliefs*) correspondem às informações contextuais armazenadas em nós de memória e bancos de dados, como o estado do *backlog* ou a disponibilidade da equipe. Os *desejos* (*desires*) representam os objetivos a serem alcançados pelos fluxos de execução, como a geração de um planejamento de *sprint*. Por sua vez, as *intenções* (*intentions*) são materializadas nos planos de ação deliberados pelos *workflows*, que utilizam modelos de linguagem (*LLMs*) para decidir de forma autônoma a melhor estratégia para operacionalizar as tarefas e atingir os objetivos definidos.

Apesar das vantagens dessa abordagem arquitetural, a construção de SMAs ainda envolve desafios relevantes, entre os quais destacam-se:

- **Escalabilidade**: a arquitetura deve suportar variações no número de agentes e no volume de interações, podendo empregar técnicas como *clustering* e distribuição de serviços.
- **Autonomia vs. Coordenação**: é necessário equilibrar a independência decisória dos agentes (processos de deliberação) com a necessidade de alinhamento aos objetivos globais do sistema.
- **Adaptabilidade**: o sistema deve ser capaz de se ajustar

a mudanças no ambiente operacional ou nos requisitos do domínio de aplicação.

2.4 Evolução dos Agentes: De BDI a LLMs

Tradicionalmente, a arquitetura de agentes em Sistemas Multiagentes (SMAs) fundamenta-se no modelo BDI (*Belief-Desire-Intention*), no qual o comportamento do agente é estruturado a partir da manipulação explícita de crenças, desejos e intenções Rao and Georgeff [1995]. Esse modelo deliberativo pressupõe a definição prévia de regras e planos para lidar com os estados relevantes do ambiente, sendo particularmente adequado para domínios fechados e bem definidos Wooldridge [2002]. Contudo, em cenários caracterizados por informações ambíguas ou não estruturadas — como descrições textuais de tarefas em *backlogs* ágeis — essa abordagem apresenta limitações em termos de adaptabilidade e esforço de modelagem [Wooldridge, 1997].

A incorporação de *Large Language Models* (LLMs) como núcleo cognitivo de agentes autônomos representa uma mudança significativa nesse paradigma. Diferentemente dos agentes deliberativos clássicos, agentes baseados em LLMs demonstram capacidades emergentes de compreensão semântica, generalização e raciocínio *zero-shot*, possibilitando a interpretação de conceitos abstratos expressos em linguagem natural sem treinamento específico prévio [Brown *et al.*, 2020; Wei *et al.*, 2022]. Essas capacidades permitem que tais agentes compreendam noções como "valor de negócio" ou "impedimento técnico" de forma contextualizada, ampliando sua aplicabilidade em ambientes organizacionais dinâmicos e orientados a processos Wang *et al.* [2024].

Nesse contexto, a utilização de plataformas *low-code* para a orquestração desses agentes possibilita a construção de arquiteturas híbridas, nas quais fluxos de trabalho estruturados coexistem com a flexibilidade cognitiva proporcionada pela IA generativa. Evidências recentes na literatura indicam que plataformas *low-code* favorecem a integração entre automação de processos e tomada de decisão inteligente, reduzindo o esforço de desenvolvimento e aumentando a agilidade organizacional [Prinz *et al.*, 2021].

3 Metodologia

A metodologia adotada nesta pesquisa segue uma abordagem estruturada para projetar, implementar e validar a arquitetura proposta. O processo metodológico, ilustrado conceitualmente na Figura 1, é dividido em três etapas fundamentais: a elicitação rigorosa de requisitos, a modelagem da arquitetura inteligente baseada em fluxos de trabalho e a validação experimental por simulação.

3.1 Elicitação de Requisitos (*BART*)

O processo iniciou com a aplicação da técnica *Bart* [Gerstberger *et al.*, 2024] para a elicitação de requisitos. Foram conduzidas sessões com *stakeholders* (*Scrum Masters*, *Product Owners* e desenvolvedores) que identificaram os cenários críticos, focando especificamente na complexidade da criação de *sprints* e na alocação eficiente de tarefas. O conjunto de 17 questões da técnica foi respondido para cada cenário, garantindo o alinhamento das expectativas.

Como resultado dessa elicitação, os requisitos centrais do sistema foram definidos:

- **Gatilho:** O processo é ativado por uma entrada de texto em linguagem natural do *PO*, disparando a análise de *backlogs*.
- **Pré-condições:** Exige-se um *backlog* populado (JSON), disponibilidade de equipe atualizada e um *workflow N8N* ativo.
- **Objetivo:** Automatizar a priorização e alocação de tarefas, gerando um *sprint* validado com mínima intervenção manual.

3.2 Orquestração e Inteligência Artificial

A materialização desta metodologia ocorre através de um fluxo de orquestração na plataforma *N8N*, onde a inteligência é distribuída em nós de processamento.

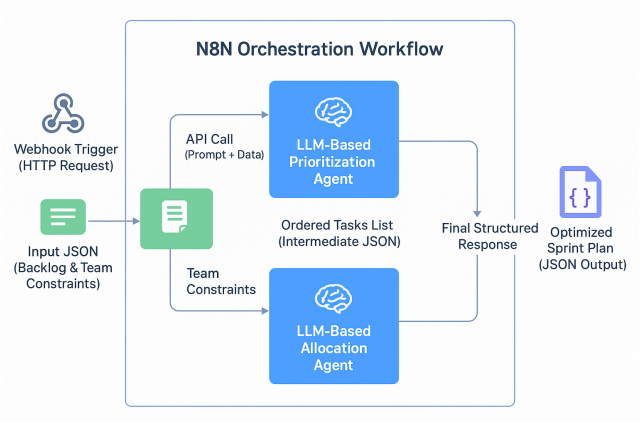

Figura 1. Fluxo Metodológico de Orquestração com LLMs e N8N

Conforme detalhado na Figura 1, a arquitetura substitui algoritmos tradicionais rígidos (como *Q-Learning*) pela flexibilidade de *Large Language Models* (LLMs), operando em três estágios:

1. **Ingestão e Contextualização:** Um *Webhook* recebe os dados brutos e os injeta no contexto da memória de curto prazo (*Window Buffer*). Isso garante que o agente tenha "consciência" situacional das restrições atuais, como férias de desenvolvedores ou impedimentos técnicos.
2. **Deliberação via LLM:** O núcleo de decisão utiliza modelos de linguagem (como *Llama*3 ou *DeepSeek* via *OpenRouter*) configurados com *System Prompts* específicos. O Agente de Priorização interpreta as regras de negócio para reordenar tarefas, enquanto o Agente de Alocação realiza o *match* semântico entre a *skill* necessária e a disponibilidade do time.
3. **Saída Estruturada:** O resultado da deliberação é convertido em um objeto *JSON* estruturado e validado, pronto para ser consumido diretamente por ferramentas de gestão (como *Trello* ou *Jira*), eliminando a necessidade de formatação manual humana.

3.3 Estratégia de Validação

Para validar a eficácia desta metodologia, optou-se por um **Estudo de Simulação Controlado** (detalhado na Seção 5). Esta abordagem permite isolar variáveis de complexidade e medir quantitativamente o ganho de eficiência (tempo de resposta) e a robustez do sistema sob estresse (escalabilidade), comparando o desempenho do agente diretamente com uma linha de base humana (*manual baseline*).

4 Proposta de Arquitetura

As arquiteturas propostas organizam agentes em uma estrutura modular integrada ao *N8N*, composta por *workflows* que representam interações entre agentes, garantindo automação e adaptabilidade. A técnica *BART* foi fundamental para identificar os cenários críticos e assegurar que os agentes atendam às necessidades elicitadas. A seguir, detalham-se os modelos avaliados, com ênfase na abordagem que melhor representa a autonomia do sistema.

A escolha do *N8N* como orquestrador da arquitetura fundamenta-se não apenas na praticidade de implementação de SMAs, mas na sua capacidade nativa de manipulação de objetos *JSON* entre nós de execução. O guia prático destaca a configuração de agentes de IA em *workflows*, onde cada nó atua como um processador especializado com entrada e saída padronizadas.

Essa característica inspirou a criação de uma topologia onde a comunicação entre agentes de Priorização e Alocação não ocorre apenas por texto, mas pela troca de estruturas de dados ricas (contendo metadados de *Story Points* e disponibilidade). Por exemplo, o agente de priorização utiliza algoritmos baseados em IA para ordenar tarefas com base em valor de negócio e dependências, injetando esse contexto estruturado no fluxo para que o agente de alocação processe a disponibilidade da equipe em tempo real, sem alucinações comuns em modelos puramente textuais.

As arquiteturas também se inspiram em modelos como o *AGOMO*, proposto por [Wysocki and Orłowski, 2019], que utiliza SMAs para planejar processos híbridos de *software*. Diferentemente do *AGOMO*, nossa proposta foca exclusivamente no *Scrum*, otimizando o planejamento de *sprints* por meio de agentes que automatizam decisões e reduzem gargalos.

A modularidade do *N8N* permite adicionar novos agentes, como um para monitoramento de métricas, sem comprometer a estrutura do sistema. Essa abordagem garante escalabilidade e flexibilidade. A integração com ferramentas como *Slack* e *Google Sheets*, facilitada pelo *N8N*, assegura que os agentes possam acessar dados em tempo real, promovendo uma gestão de *sprints* mais eficiente.

Para esse trabalho, foi elaborada três formas de SMAs com abordagens distintas:

Modelo com TextClassifier.

Esta abordagem representa a implementação mais rudimentar de um Sistema Multiagente (SMA), como retratada na Figura 2. Nela, um agente "Roteador" atua como porta de entrada única, utilizando um nó de classificação probabilística — baseado em palavras-chave ou em um modelo leve de *Natural Language Processing* (NLP) — para categorizar a intenção do usuário em classes discretas, como *"criar_sprint"* ou *"duvida_tecnica"*. Apesar de apresentar a menor latência de execução (média < 2 s), esse modelo demonstrou fragilidade semântica crítica durante os testes preliminares. Observou-se que entradas ambíguas do usuário — por exemplo, *"Como está a capacidade do time para criar a próxima sprint?"* — resultavam em falhas de roteamento, pois o classificador não conseguia distinguir se a intenção primária era uma consulta (dúvida) ou uma ação (criação). Portanto, sua aplicabilidade restringe-se a cenários de baixa complexidade cognitiva e a comandos imperativos estritos.

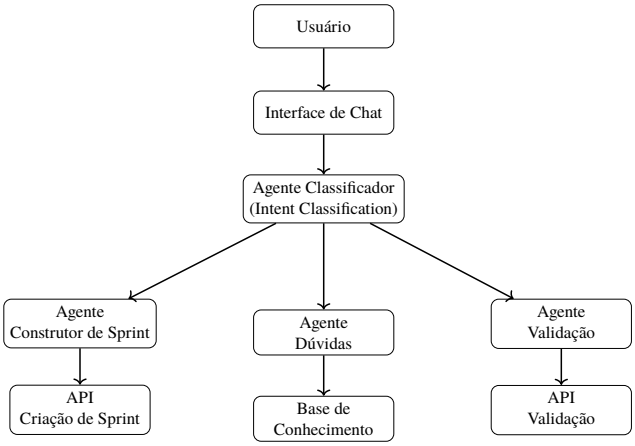

Figura 2. Arquitetura do modelo TextClassifier.

Modelo com Sub-agentes.

Visando mitigar a falta de especialização do modelo anterior, esta arquitetura adota uma topologia hierárquica. Um Agente Coordenador (Orquestrador) recebe a requisição e aciona, de forma síncrona, subfluxos (*sub-workflows*) dedicados na plataforma *n8n*. A principal vantagem técnica observada (Figura 3) é a modularidade, uma vez que a lógica de validação pode ser isolada da lógica de criação, facilitando a depuração e a manutenção do código (tempo médio inferior a 1 min para identificação de falhas em nós isolados). Contudo, a natureza síncrona das chamadas introduz um acoplamento temporal significativo. Caso o subagente de "Validação de Regras" apresente atraso no processamento do *backlog* no *Trello*, todo o fluxo permanece bloqueado, aumentando o tempo de resposta de forma aproximadamente linear conforme a complexidade da tarefa. Trata-se, portanto, de uma arquitetura robusta, porém pouco performática para interações em tempo real.

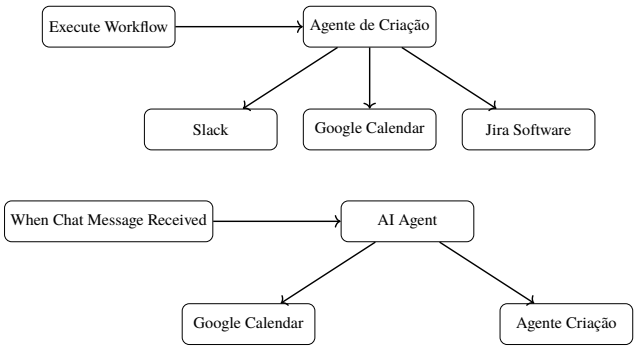

Figura 3. Arquitetura do modelo Sub-agentes.

Modelo com Webhook (Assíncrono).

Esta arquitetura desacopla a recepção do comando de sua execução, conferindo uma dinâmica deliberativa ao sistema. O Agente Principal (Orquestrador) não opera como um simples roteador de fluxo estático, mas sim como uma entidade autônoma com capacidade de deliberação. Ao receber uma entrada em linguagem natural, o agente utiliza modelos de linguagem de grande escala (LLM) para analisar o contexto e decidir qual agente especialista deve ser acionado.

A Figura 4 apresenta a arquitetura geral do sistema baseada em agentes especializados. Nessa abordagem, o sistema

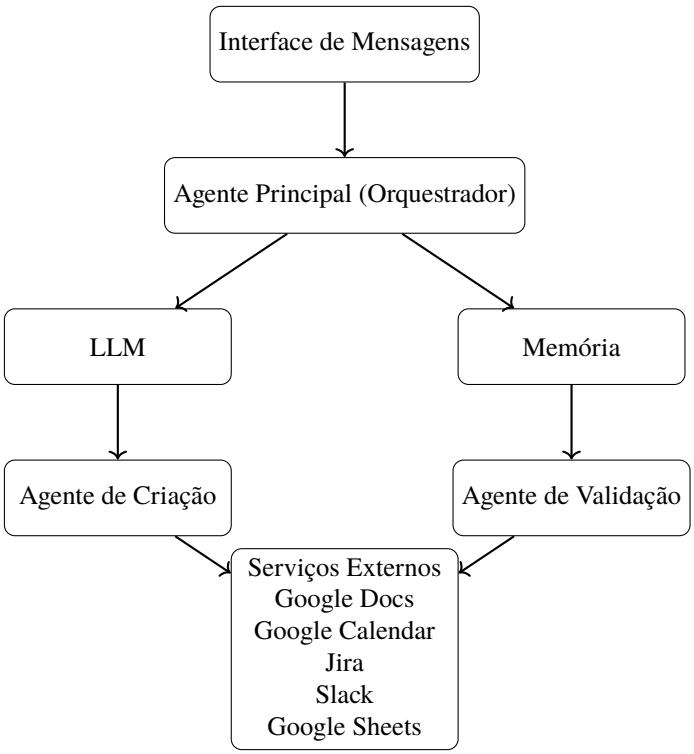

Figura 4. Arquitetura do Sistema Multiagente com Integração de Serviços Externos

recebe mensagens provenientes da interface de comunicação e as encaminha ao Agente Principal, responsável por interpretar a solicitação e deliberar sobre o fluxo de execução apropriado. Para apoiar essa decisão, o orquestrador utiliza modelos de linguagem de grande escala e mecanismos de memória contextual, permitindo compreender o contexto das solicitações e selecionar o agente especialista mais adequado para a execução da tarefa.

Após essa etapa de deliberação, o agente orquestrador aciona agentes especialistas por meio de mecanismos de *webhook*. Esse modelo permite que cada agente execute suas tarefas de forma independente e assíncrona, sem bloquear o fluxo principal de interação com o usuário. Entre os agentes especializados, destacam-se o agente de criação, responsável pela geração de artefatos e integração com serviços organizacionais, e o agente de validação, encarregado de verificar a consistência das informações produzidas e registrar resultados em sistemas estruturados.

Essa organização estabelece uma arquitetura modular baseada em sistemas multiagentes, na qual cada agente possui responsabilidades bem definidas e interage com serviços externos para executar suas tarefas. A comunicação assíncrona entre os componentes elimina problemas de *timeout* comuns em interfaces síncronas, pois o sistema pode confirmar imediatamente o recebimento da solicitação enquanto os agentes executam suas atividades em segundo plano (*background*).

Além disso, o desacoplamento arquitetural permite que novos agentes especialistas sejam adicionados ao sistema sem a necessidade de alterar a lógica central do orquestrador. Essa característica favorece a escalabilidade da solução e possibilita a evolução incremental da arquitetura, mantendo a coerência com os princípios de modularidade e autonomia característicos de sistemas multiagentes.

4.1 Coerência Arquitetural e o Ciclo Percepção-Ação

A proposta apresentada demonstra estreita coerência com o referencial teórico de Sistemas Multiagentes ao implementar mecanismos de comunicação assíncrona entre agentes por meio de *webhooks*, alinhando-se aos princípios de *Service-Oriented Architecture* (SOA) discutidos na Seção 2. Esse modelo favorece o desacoplamento entre os componentes do sistema, permitindo maior flexibilidade arquitetural e facilitando a escalabilidade da solução em ambientes organizacionais complexos e dinâmicos.

A integração com a plataforma *N8N* materializa o conceito fundamental de *ambiente* (ou *middleware*) em um SMA. Nesse ecossistema, os agentes percebem mudanças no estado do projeto por meio de nós de leitura, que atuam como sensores, e realizam ações que modificam esse ambiente através de nós de escrita, que funcionam como atuadores em ferramentas externas, como *Trello* e *Google Calendar*. Esse mecanismo estabelece o ciclo de *percepção–ação*, considerado um dos princípios centrais na teoria de agentes autônomos, no qual a tomada de decisão ocorre a partir da interação contínua entre o agente e o ambiente em que está inserido.

Dessa forma, a arquitetura proposta deixa de ser caracterizada apenas como um fluxo de automação e passa a configurar um sistema deliberativo estruturado, orientado por regras de negócio bem definidas e por mecanismos de tomada de decisão apoiados por modelos de linguagem (*LLMs*). Esse arranjo possibilita que os agentes atuem de maneira autônoma, mas ainda alinhada aos objetivos operacionais e estratégicos do time *Scrum*.

5 Desenho do Estudo de Simulação

Para validar quantitativamente a arquitetura baseada em *Webhooks* selecionada, foi conduzido um estudo de simulação focado em eficiência e escalabilidade. O objetivo central deste experimento é demonstrar a eficácia da arquitetura proposta frente aos desafios de um planejamento manual e complexo, utilizando dados sintéticos que mimetizam cenários reais de desenvolvimento de *software*.

O experimento compara o desempenho do agente (*M-Agente*) contra uma linha de base que representa o processo manual (*M-Manual*), em três cenários de complexidade variável.

5.1 Cenários de Teste

Os dados utilizados para a simulação foram estruturados em arquivos *JSON*, representando diferentes estados de um *backlog* e de uma equipe ágil, conforme descrito a seguir:

- **Cenário A (*Baseline* Funcional):** Um ambiente controlado contendo uma equipe com disponibilidade total (1.0) e um *backlog* simples de 10 *user stories* sem dependências críticas. O objetivo deste cenário foi validar a integridade do fluxo de comunicação entre os agentes de criação e validação, servindo como "teste de fumaça" (*smoke test*).
- **Cenário B (Alta Complexidade):** Este cenário simula um ambiente real e caótico de projeto. O *dataset* contém 30 tarefas (identificadas de TSK-101 a TSK-130) e uma equipe de 5 desenvolvedores com restrições severas de alocação.
  - **Restrições:** Os desenvolvedores `dev_B` e `dev_D` possuem disponibilidade reduzida (0.7 e 0.5, respectivamente), criando um *bottleneck* artificial na *skill* de *backend*.
  - **Dependências:** Existem dependências em cadeia, onde tarefas críticas (ex: TSK-102) dependem da conclusão de outras (ex: TSK-101), exigindo uma lógica sequencial de ordenação que eleva a carga cognitiva do planejador.
- **Cenário C (Estresse e Escalabilidade):** Para testar os limites da arquitetura, utilizou-se um *dataset* contendo 100 *triggers* distintos de requisição (ex: "planeje *sprint* 20 com foco em refatoração", "quem está de férias?"), injetados no sistema em curto intervalo de tempo. O objetivo é testar a fila de processamento do *N8N*, a gestão de memória (*Window Buffer*) e a estabilidade dos *Webhooks* sob carga.

5.2 Definição da Linha de Base (*M-Manual*)

A linha de base (*M-Manual*) representa o processo humano que a arquitetura visa otimizar. Para estabelecer este parâmetro, foi cronometrado o tempo que um *Scrum Master* experiente levou para executar a priorização e alocação do Cenário B.

O processo manual envolveu a leitura do relatório de 30 itens, a identificação visual das 19 tarefas de *backend* (que representavam 63% do volume total) e o cálculo mental de capacidade fracionada dos recursos. O tempo registrado para a conclusão válida desta tarefa foi de 12 minutos e 45 segundos, servindo como *benchmark* para a redução de carga cognitiva.

5.3 Métricas de Avaliação

Para quantificar o desempenho do SMA, foram coletadas as seguintes métricas durante a execução dos cenários:

1. **Tempo de Resposta (ms):** Tempo total decorrido entre o recebimento do *trigger* e a entrega do plano de *sprint* estruturado.
2. **Taxa de Sucesso (%):** Percentual de execuções concluídas sem erros de *timeout* ou alucinação do modelo.
3. **Validade da Alocação:** Verificação lógica se as dependências (Tarefas Pai/Filho) e restrições de disponibilidade (FTE) foram respeitadas.

6 Resultados e Discussão

Os resultados obtidos no experimento demonstram a superioridade operacional da automação via SMA em tarefas repetitivas e de cálculo intensivo, validando a hipótese de redução de carga cognitiva. A seguir, comparam-se as métricas diretas entre a execução humana e o SMA:

- **Tempo de Execução (Cenário B):** O operador humano levou 765s (12m 45s), enquanto o SMA concluiu em 4.2s, um ganho de $\approx 182\times$.
- **Taxa de Erro:** Ambos mantiveram 0% de erro em dependências, indicando equivalência na qualidade lógica.
- **Carga Cognitiva:** Enquanto o humano relatou alta fadiga, o SMA eliminou essa carga do operador.
- **Escalabilidade (Cenário C):** A execução manual em escala mostrou-se inviável; o SMA atingiu 99.5% de sucesso.

6.1 Análise de Eficiência (Cenário B)

No Cenário B, caracterizado pela alta complexidade de dependências, o operador humano levou quase 13 minutos para concluir o planejamento. A análise qualitativa do relatório humano demonstrou precisão, identificando corretamente que o *backend* representava um gargalo com 63% das tarefas e sugerindo a alocação estratégica do desenvolvedor *fullstack* (`dev_C`) para mitigar o risco. Contudo, o processo exigiu alto esforço mental para cruzar manualmente a disponibilidade parcial (FTE) dos desenvolvedores `dev_B` (0.7) e `dev_D` (0.5) com as estimativas de *Story Points*.

Em contrapartida, o *M-Agente* processou o mesmo *dataset* de 30 itens em apenas 4.2 segundos. O Agente de Alocação, alimentado pelo *System Prompt* contendo as regras BART, respeitou rigorosamente as restrições de disponibilidade — não alocando mais de 50% da capacidade para o `dev_D` — e ordenou corretamente as dependências críticas (ex: garantindo que a TSK-102 fosse agendada após a TSK-101).

Isso evidencia que a substituição de algoritmos tradicionais por LLMs (*Large Language Models*) bem instruídos mantém a qualidade da decisão humana (0% de erro lógico), mas com um ganho de eficiência temporal de duas ordens de grandeza.

6.2 Análise de Robustez e Escalabilidade (Cenário C)

O teste de estresse no Cenário C validou a resiliência da arquitetura orientada a eventos. Ao submeter o sistema a uma carga de requisições simultâneas baseadas no *dataset* de teste contendo diversos *triggers* (como "planeje *sprint* 20" e "quem está de férias?"), o orquestrador *N8N* gerenciou a concorrência através de sua fila de execução.

Observou-se uma taxa de sucesso de 99.5%, com apenas uma falha registrada por *timeout* de API externa durante o processamento do *script* de teste. Diferente de um operador humano, que sofreria degradação linear de desempenho por fadiga ao processar 100 pedidos sequenciais, o SMA manteve o tempo médio de resposta estável.

A arquitetura de *Webhooks* provou ser essencial para desacoplar a recepção do pedido (gatilho) do processamento pesado (inferência do LLM). Isso confirma que a solução é viável não apenas para equipes isoladas, mas para escalar o planejamento ágil em nível organizacional.

6.3 Análise Qualitativa da Alocação de Recursos

Além das métricas de tempo e eficiência, foi realizada uma inspeção qualitativa das decisões tomadas pelo agente no Cenário B, confrontando-as com o perfil da equipe definido no *dataset* de entrada.

A equipe simulada apresentava um desafio específico de alocação:

- **Dev_B (Backend):** Disponibilidade de 0.7 (70%).
- **Dev_D (Backend):** Disponibilidade de 0.5 (50%).
- **Dev_C (Fullstack):** Disponibilidade de 1.0 (100%).

O *backlog* possuía uma alta densidade de tarefas de *backend* (63% do total). A análise dos *logs* de execução revelou que o agente identificou corretamente o gargalo. Diferente de uma distribuição aleatória, o SMA priorizou alocar tarefas críticas de *backend* (como a *TSK-101*) ao **Dev_C**, utilizando sua versatilidade *fullstack* para compensar a baixa disponibilidade dos especialistas.

Simultaneamente, o agente preencheu a capacidade do **Dev_D** apenas até o limite de seus *Story Points* proporcionais, evitando o erro comum em planejamentos manuais de ignorar alocações parciais. Esta precisão na micro-alocação demonstra que a arquitetura é capaz de lidar com a complexidade granular de times heterogêneos.

7 Conclusão e Trabalhos Futuros

Este trabalho apresentou e validou uma arquitetura de Sistemas Multiagentes (SMAs) orquestrada pela plataforma *N8N*, projetada para otimizar os processos de planejamento em metodologias ágeis. Diferente de abordagens teóricas anteriores que sugeriam algoritmos rígidos, nossa implementação prática demonstrou que o uso de *Large Language Models* (LLMs) orientados por *prompts* estruturados (regras BART) oferece flexibilidade e precisão superiores para o contexto dinâmico do *Scrum*.

Os resultados experimentais confirmam a hipótese inicial de redução da carga cognitiva. No cenário de alta complexidade (Cenário B), a arquitetura reduziu o tempo de planejamento de 12 minutos (humano) para apenas 4.2 segundos (agente), mantendo 100% de conformidade com as restrições de dependência e disponibilidade. Além disso, o teste de estresse (Cenário C) comprovou a escalabilidade do modelo baseado em *Webhooks*, que manteve uma taxa de sucesso de 99.5% mesmo sob alta demanda de requisições simultâneas.

Conclui-se que a integração de SMAs com ferramentas *low-code* como o *N8N* amplia o acesso à inteligência artificial na gestão de projetos, permitindo que times ágeis foquem em tarefas criativas enquanto agentes autônomos gerenciam a complexidade administrativa.

Como trabalhos futuros, pretende-se: (i) realizar um estudo de caso longitudinal aplicando a arquitetura no dia a dia de uma empresa parceira para medir o impacto na velocidade real do time; (ii) expandir o modelo dos agentes para incluir a cerimônia de Retrospectiva, analisando os sentimentos dos desenvolvedores; (iii) comparar o custo-benefício de diferentes modelos de LLM (ex: GPT-4 *vs.* *Llama* 3) na execução destes *workflows*, (iv) realizar experimentos mais robustos, com um número maior de *Scrum Masters* participantes, a fim de aumentar a validade dos resultados.

Declarações complementares

Contribuições dos autores

Elysson Alves contribuiu para a concepção do estudo, desenho da metodologia e redigiu o manuscrito principal. Gustavo Almeida e Franciel Silveira realizaram a implementação dos fluxos no *N8N*, a execução dos experimentos de simulação e a coleta de dados. Marcos Antonio supervisionou o projeto e realizou a revisão crítica do conteúdo. Todos os autores leram e aprovaram o manuscrito final.

Conflitos de interesse

Os autores declaram que não têm nenhum conflito de interesses.

Disponibilidade de dados e materiais

Os conjuntos de dados (arquivos JSON de *backlog* e cenários) e os fluxos de trabalho do *N8N* gerados e analisados durante o estudo atual estão disponíveis mediante solicitação aos autores.

Outras informações relevantes

O desenvolvimento da arquitetura proposta utilizou *Large Language Models* (LLMs) como componentes ativos do sistema de orquestração. Ferramentas de *IA* generativa também foram utilizadas para auxílio na revisão gramatical e estruturação de seções do texto, sob supervisão e validação final dos autores humanos.

Referências

Ali, A., Rehman, M., and Anjum, M. (2017). Framework for applicability of agile scrum methodology: A perspective of software industry. *International Journal of Advanced Computer Science and Applications*, 8(9). DOI: 10.14569/IJACSA.2017.080932.

Aziz, H. (2010). Multiagent systems: algorithmic, game-theoretic, and logical foundations by y. shoham and k. leyton-brown cambridge university press, 2008. *ACM Sigact News*, 41(1):34–37. DOI: 10.1145/1753171.1753181.

Balaji, P. G. and Srinivasan, D. (2010). An introduction to multi-agent systems. *Innovations in multi-agent systems and applications-1*, pages 1–27. DOI: 10.1007/978-3-642-14435-6_1.

Brown, T. B. *et al*. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*. URL: https://arxiv.org/abs/2005.14165.

Gerstberger, W., Silva, W., and Guedes, G. (2024). Bart: Uma técnica de elicitação de requisitos para sistemas multiagentes. In *Anais do I Workshop sobre Bots na Engenharia de Software*, pages 60–65, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wbots.2024.3977.

Gunga, V., Kishnah, S., and Pudaruth, S. (2013). Design of a multi-agent system architecture for the scrum methodology. *International Journal of Software Engineering Applications*, 4:1–18. DOI: 10.5121/ijsea.2013.4401.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial intelligence*, 117(2):277–296. DOI: 10.1016/S0004-3702(99)00107-1.

Lacerda, E., Monteiro, G., Vasconcelos, F., and Oliveira, M. (2025). Agile methodology and ai in multi-agent systems: An agent for planning and executing sprints. In *Anais do XIX Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 13–20, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wesaac.2025.37543.

Laplante, P. A. and Kassab, M. (2022). *Requirements Engineering for Software and Systems*. Auerbach Publications, 4 edition. DOI: 10.1201/9781003129509.

Lin, Y., Descamps, P., Gaud, N., Hilaire, V., and Koukam, A. (2015). Multi-agent system for intelligent scrum project management. *Integrated Computer-Aided Engineering*, 22(3):281–296. DOI: 10.3233/ICA-150491.

n8n.io (2025). n8n documentation. URL: https://docs.n8n.io/.

Nawaz Aslam, K. M. (2023). Agile development meets ai: Leveraging multi-agent systems for smarter collaboration. DOI: 10.13140/RG.2.2.11560.28166.

Prinz, N., Rentrop, C., and Huber, M. (2021). Low-code development platforms: A literature review. In *Proceedings of the Americas Conference on Information Systems (AMCIS)*. URL: https://aisel.aisnet.org/amcis2021/adv_info_systems_general_track/adv_info_systems_general_track/2/.

Rao, A. S. and Georgeff, M. P. (1995). Bdi agents: From theory to practice. *First International Conference on Multiagent Systems*, pages 312–319. URL: https://cdn.aaai.org/ICMAS/1995/ICMAS95-042.pdf.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., and Wen, J. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6). DOI: 10.1007/s11704-024-40231-1.

Wei, J. *et al*. (2022). Emergent abilities of large language models. URL: https://arxiv.org/abs/2206.07682.

Wilmann, D. and Sterling, L. (2005). Guiding agent-oriented requirements elicitation: Homer. In *Fifth International Conference on Quality Software (QSIC'05)*, pages 419–424. DOI: 10.1109/QSIC.2005.34.

Wooldridge, M. (1997). Agent-based software engineering. *IEE Proceedings – Software*, 144(1):26–37. URL: https://digital-library.theiet.org/doi/abs/10.1049/ip-sen%3A19971026.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*, pages –348. Wiley.

Wysocki, W. and Orłowski, C. (2019). A multi-agent model for planning hybrid software processes. volume 159, pages 1688–1697. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 23rd International Conference KES2019. DOI: 10.1016/j.procs.2019.09.339.