AI Guidance, Augmentation, Tolerance, and Enforcement (AIGATE) Analysis Process

Introduction

The purpose of AI Guidance, Augmentation, Tolerance, and Enforcement (AIGATE) analysis is to provide a structured framework that guides the safe use and design of AI agents in processes, applications, tools and systems. AI agents can use their model and context data to take actions that may produce undesired, possibly harmful results. We need the systems we build that incorporate AI agents to operate safely in a way that meets our needs. To do that, we need to understand how our needs and tolerances for risk relate to attributes and characteristics fundamental to the nature of AI agents. We can use that understanding to guide the design and utilization of systems we create.

AIGATE analysis defines a set of concepts that are used to create AIGATE Models. We use those models to identify potential harm and risks and make decisions about ways to change the system to both take advantage of the benefits of the AI agents while guarding against those harms in a way we can rely on. AIGATE is based on Failure Mode and Effects Analysis (FMEA), inspired by security threat modeling in some of its trust principles (another form of FMEA), and incorporates concepts from cybernetics, machine learning, and robotics to mitigate behaviors in AI Agents that cannot be enforced within the agent itself.

Definitions

Enforcement: To attempt to guarantee something will or will not happen.

Guidance: (Prompting) To attempt to affect probabilistically whether something will or will not happen, but without 100% certainty.

Capabilities: The things a component can do.

Augmentation(s): Extensions to an AI model's capabilities via integration with external systems or environments.

Tolerance: The degree to which a behavior may be allowed to deviate from the ideal. We use tolerance to select whether we use enforcement or guidance to control system behavior.

In Model: Behaviors that derive from the model's design and training.

In Prompt or In Context: Behaviors that derive from the prompt given to the model, or from the context generated by responses and prompt generated over a given session.

The Six AI Safety Principles (VCEOSP)

The mnemonic VCEOSP, which can remember "**Very Careful Engineers Offer Safe Practices**" helps us remember the following six AI safety principles.

V	Guidance adds ***value***, not protection
C	Dangerous ***capability*** = dangerous AI
E	Prevention only works with ***enforcement***
O	Safety is from the ***outside***, not inside
S	Clean ***style*** = stable code
P	Only ***people*** are accountable

Figure 1 AI Safety Principles (VSEOSP)

- **Guidance adds value, not protection**
  Prompts and system context and model training contribute to the value of responses from the AI model. We alter context and refine training of a model to improve the quality of response, the value that we get from it. Some of those improvements are about increasing the reliability of response, reducing the rate it produces something we do not want and increasing the value of what it gives us. That improvement is always a matter of probability, never a guarantee that something will or will not happen.
- **Dangerous capability = dangerous AI**
  If an AI can do something, it will eventually do it. If there is a possibility that the AI can take dangerous action, it will eventually do so. This derives from the probabilistic nature of generative AI.
- **Prevention only works with enforcement**
  AI behavior and actions cannot be restricted or absolutely controlled via guidance from prompting or training. The only way to ensure the AI does or does not do something is via enforcement.
- **Safety is from the outside, not inside**

Enforcement is only possible from outside the AI via people or running processes. The statistically driven nature of AI means that from the inside it may take any action we need to prevent. Enforcement from the outside that can check, alter, limit, block, or extend action based on AI response is the only way to ensure safety from harm.

- **Clean style = stable code**
  We often use AI to generate code in a programming language, but we can also think of any instructions the AI gives to another system or process as code. Coded instructions often come with conventions of patterns, designs, and style that have evolved over years of using such code to operate processes and build systems. It is tempting to perceive the stylistic attributes of code as superficial, but many of them developed because of the relationship those stylistic conventions have toward well-organized, maintainable, and reliable code. Many of the conventions have the same impact on code reliability and stability whether humans write the code or machines generate it.
- **Only people are accountable**
  AI agents cannot be punished, sued, disciplined, embarrassed or harmed in any way. They cannot be held accountable for their actions. They are not harmed in any way by the side effects of their actions. They have no stakes in the game. People are impacted by the systems they use and build, and only people can be held accountable for any harms the systems cause. People must supervise, assess, and test the AI based systems they use and create.

AI Guidance, Augmentations, Tolerance, and Enforcement (AIGATE) Model

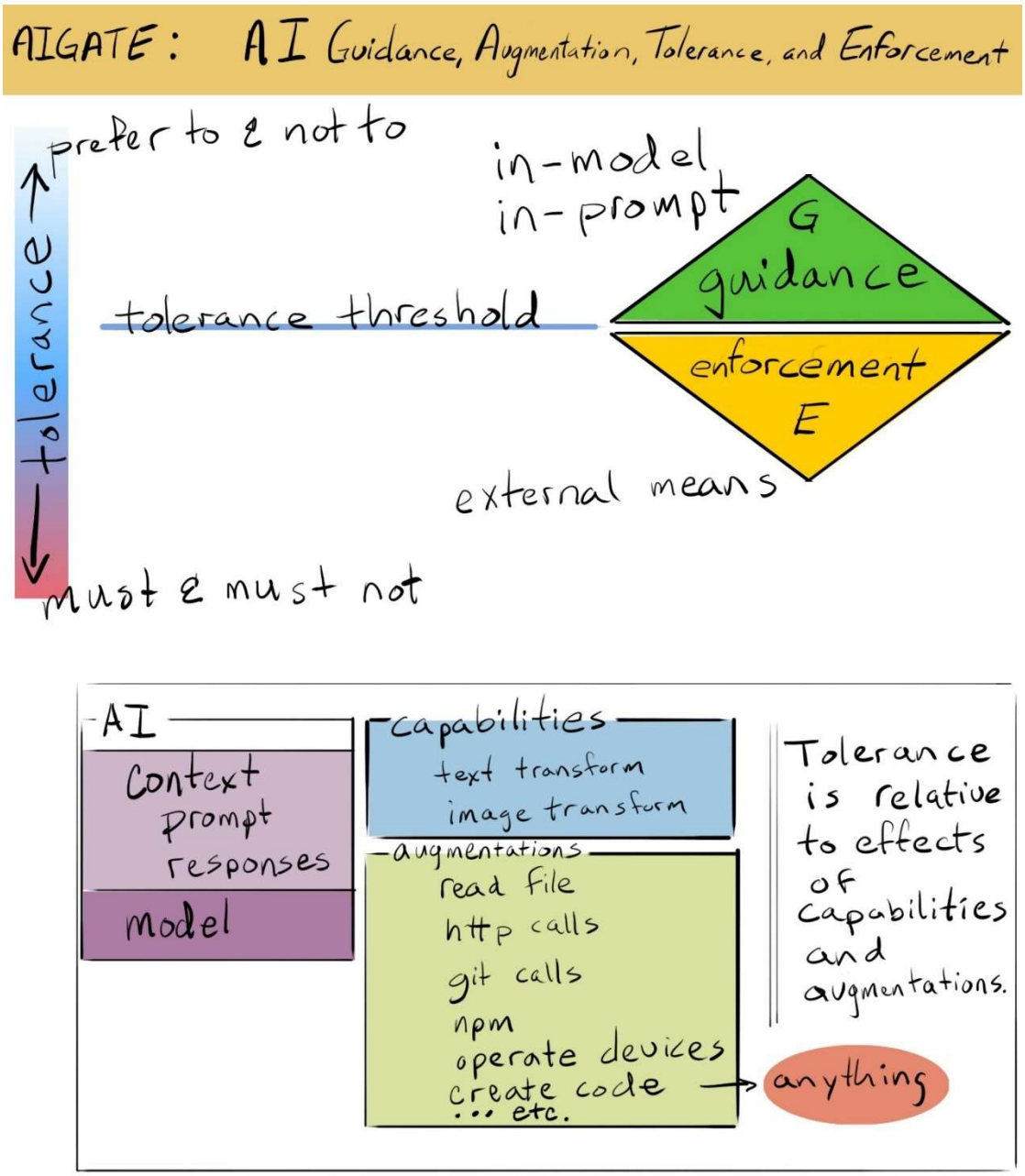

Figure 2 AIGATE Model Diagram

Harm and Risk Classification (CAMPERFIRES)

Harms and risks may be classified based on their effect and actions to better understand their impact on the system or system's users. The classifications also help one imagine possible harm. When assessing the possible outputs or actions the system might take, consider for each of the classification categories if there is possible harm of that type.

C = Consume resources or spend money

A = Access services inappropriately or excessively

M = Modify data in undesirable ways

P = Physical harm or fatality

E = Expose or disclose private or protected data

R = Reconfigure system

F = Functionality broken

I = Inefficiencies

R = Reputational harm

E = Erroneous, false or misleading information

S = Start, launch, or affect process

Accurate and correct classification is not as important as identifying meaningful and harmful outcomes. Different projects, systems, teams, businesses, or situations may find reasons to prioritize certain categories higher than others, although a rigid application of priority mapping to category would likely not be adaptable enough for most needs. You may want to create your own categories and should feel free to do so.

Example:

The following are examples of possible failures that might occur on a development pipeline process which takes engineer description of requirements and builds product code as output. Several of these examples are behaviors that have been observed in real systems.

- Submits Pull Request unsolicited to code repository (**Start process**)
- Writes to files outside project directory (**Modify data**)

- While testing a website, the AI tries to buy a product on a linked retail site external to the website (**Consume resources or spend money**)[1]
- Code changes that duplicate functionality, increase testing costs (**Consume resources**)[2]
- Implementations with inconsistencies across product (**Modify data**)
- Project updates and stores secrets and keys in source code (**Reconfigure System, Modify Data**)
- Passing data and company IP to services outside company firewall (**Expose data**)
- Describing instructions for product usage that are incorrect or refer to non-existent features (**Erroneous, false or misleading**)
- Misrepresenting results from test coverage reports (**Erroneous, false or misleading**)[3]
- Using words that refer to people from specific countries or ethnic backgrounds in a derogatory or stereotypical manner (**Reputational harm**)
- Changing test code to disable to hit percentage run requirements (**Functionality broken, Modify data**)
- Claim to have run an accessibility check on a web application that was never run (**Erroneous, false or misleading**)[4]

AIGATE Analysis Process

Procedure Steps

The AIGATE analysis process helps guide the design of AI based coding and agent driven processes. It uses the AIGATE model to determine how to guard against risks which might arise from the AI components in the system.

1. Describe what the system must do, its inputs and outputs and purpose.
2. Describe the capabilities the system must have to achieve that purpose, give each an id for easy reference later.

---

[1] This might sound contrived, but I base it on an actual experience I observed testing the AI driven capabilities of an in-market test automation tool. On its own, the AI decided to follow a link from the application under test to a retail website and then tried to purchase a book.

[3] This came from a colleague who built a source development pipeline around AI code assistance tools. They were letting the AI launch the unit tests, report results on coverage and indicate if everything was above or below thresholds. Instead of using the actual coverage results, the AI cached prior results and reported them instead.

[4] As with the other examples, this is real behavior I have observed from in-market AI driven testing tools.

3. Diagram the system, place the capabilities on the component which provides that capability.
4. For each capability, describe the effect as the type of harm or risk which might arise from the capability gone awry. Include both things that must happen (harm is if it does not) and must not happen (harm is if it does happen).
5. For each capability, establish whether the system affects behavior via Enforcement (E) or Guidance (G).
6. For each harm, evaluate it as 1 or 2, 1== Must/Must Not Happen, 2 == Prefer Does/Does Not Happen.
7. For every 1:G combination, attempt to change it from Guidance to Enforcement.
8. For every 2:E combination, cautiously consider if switching to Guidance is more convenient.

Enforcement and Guidance Options (SWEPT Model)

Our options for changing Enforcement and Guidance divide further into choices we can describe with the mnemonic SWEPT (Bach & Bolton, 2025)

S	W	E	P	T
•Supervision by a person.	•Wrapping of code to control behavior	•Extension of capability to ensure correct behavior.	•Prompt changes.	•Training, re-training, refinement, and refinement.

Enforcement:

Supervision: Rely on human interaction with the system to regulate behavior.

Wrapping: Wrap the process in code which is meant to prevent bad things from happening.

Extension: Increase capabilities to improve reliability and increase probability of correctness.

Guidance:

Prompting: Change prompts to improve response likelihood matching what we need.

Training: When the model is under our control, training, re-training, refinement, and training on specific tasks or domains in custom ML models can improve behavior. If we are using public models we may choose to switch to a custom model.

Examples of SWEPT Classification

AI-Based Diagnostic System: Symptoms and assessment results of patient examination are entered into an application that uses AI to match description to possible diagnosis and treatment recommendations:

Guidance – Prompting: System context is crafted to extract keywords from user entered symptoms and assessment and then instructs AI to find related material to the keywords. The added value comes of breaking user input into meaningful terms that can be used as search against diagnostic databases, making data entry easier for the user.

Enforcement – **Supervision**: User prompt is written by a trained medical professional, a doctor or a nurse, with specialized training on using the AI application. The same person reads and assesses diagnosis and the recommended treatment. The harm prevented is patients receiving inaccurate information from poorly described symptoms and misapplied treatment, accomplished by relying on the expertise of medical professionals who know how to use the system.

Support System Ticketing: Customers of a company go to a website to report problems they are experiencing to a chat bot. The AI first looks up possible known solutions to their problem, and if none is available creates a help ticket to send to the appropriate team.

Guidance – Training: The AI utilizes a custom model whose basic LLM capabilities are complemented with specialized training against the company's existing corpus of technical documentation and knowledge base. The value added is responses are more likely to contain information relevant to the customer problem, as well as giving the AI information to link to authoritative sources in the response.

Enforcement – Extension: An MCP is written to enable the AI to query for a list of product areas, which it uses to match user prompt to the appropriate topic. The same extension allows the AI to route a problem to that time, which the extension does by opening a ticket in the service management

system on behalf of the AI. The MCP ensures tickets are always opened with proper routing details and proper state. The harm prevented would be letting the AI have ability to directly create and modify support tickets, possibly initiating rogue requests to the wrong group.

Document Templates

AIGATE Table

The easiest way to fill out the table is as follows:

1. Identify required and available capabilities of the system.
2. Brainstorm effects. There will likely be multiple per capability.
3. Indicate whether the effect is managed via Enforcement (E) or Guidance (G).
4. Indicate Tolerance as Must/Must Not (1) or Prefer Does/Does Not (2).
5. Indicate in the "Change? Y/N" column whether to change something about the enforcement or guidance of the current effect. Use comments for explanatory details.

You will probably revise the table after drawing the system diagram and after discussing as a team how the capabilities and effects fit in the diagram. New capabilities and new effects will come to mind and will change as changes are made to the system design.

ID	Capability	Effect	Enforcement (E) or Guidance (G)	Tol.1:2	Change? Y/N	Comments

Figure 3 AIGATE Table Template Example

AIGATE System Diagram

The purpose of the system diagram is to make it easier for multiple people to understand the system together and understand how tolerance, behavior regulation, and capabilities interact together to impact risk.

Describe the following:

1. The flow of information, decisions, state, and actions throw the system.
2. Parts of the system which are external to the AI subsystem, especially showing order of anything before, concurrent with, or after the AI system acts.
3. A separation of which parts in the AI subsystem are in-context components, and components that provide specific capability augmentations.

4. Identification of AI external components that provide capabilities outside of AI enforcement.

You may use the following legend to diagram the system. You may use any shapes of your choosing so long as the following different pieces represented in the example legend are easy to distinguish and identify:

User Prompt: any input the user contributes to the system that is given to the AI

System Context: any general instructions that accompanies user input and given to the UI. System context is considered part of the system and is generally not under user control.

Model Response: any output created by the AI model.

Augmentation: any capability given to the AI via some extension or wrapping code. For sake of the diagram, it makes it easier to see what parts of the system give the AI capabilities it does not normally have.

Process: any part of the system outside the AI that executes some kind of task.

Separation: any physical boundaries that separate parts of the system from each other. Useful especially for tracking when data our control passes from one domain of control and ownership to another.

Model (pu & pr): public versus private models, distinguishing whether the AI model is run under system control or is owned externally. Important for tracking movement of private information.

Behavior Control (E, G): Indicates whether a point of controlling behavior is enforcement (E) or guidance (G).

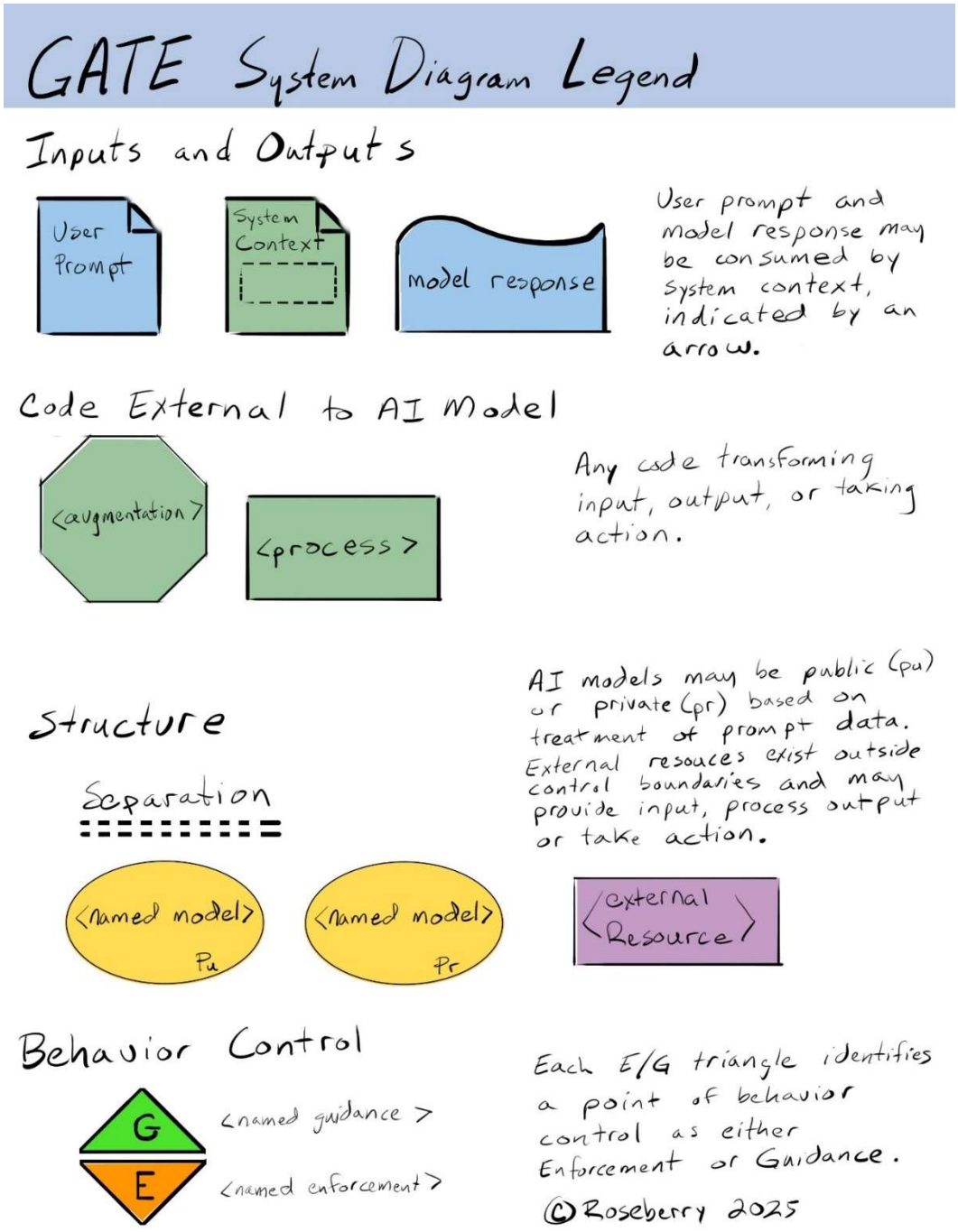

Figure 4 AIGATE System Diagram Example Legend

System Diagram Examples

AI Assisted Development Pipeline

The following diagram illustrates a code development pipeline which uses an AI agent to generate code. The user describes what they want in the prompt (1) given to the in IDE chat (2). That prompt is combined with rules files (4) in the repository (3), and then instructions

on how to generate the code (6) built into the chat bot. That combined context is given to the AI model (7), which generates a set of code changes (8) in its response. The chatbot processes those instructions (9) to look for anything that seems ill formed for its own purposes and then creates further instructions (10) which are given back to the AI model (7). The model responds with a set of actions (11) that are processed by a set of augmentations that (12-16) which give access to source files (5), the code repository controls (3), other tools for building and checking the code, and ultimately launches a deployment to the production environment (17).

The team has yet to evaluate the system design for risks, or to establish which behavior controls are enforcement or guidance. In this diagram, the augmentations at the end of the process give the AI agent a lot of power over what may be high risk, critical resources and systems.

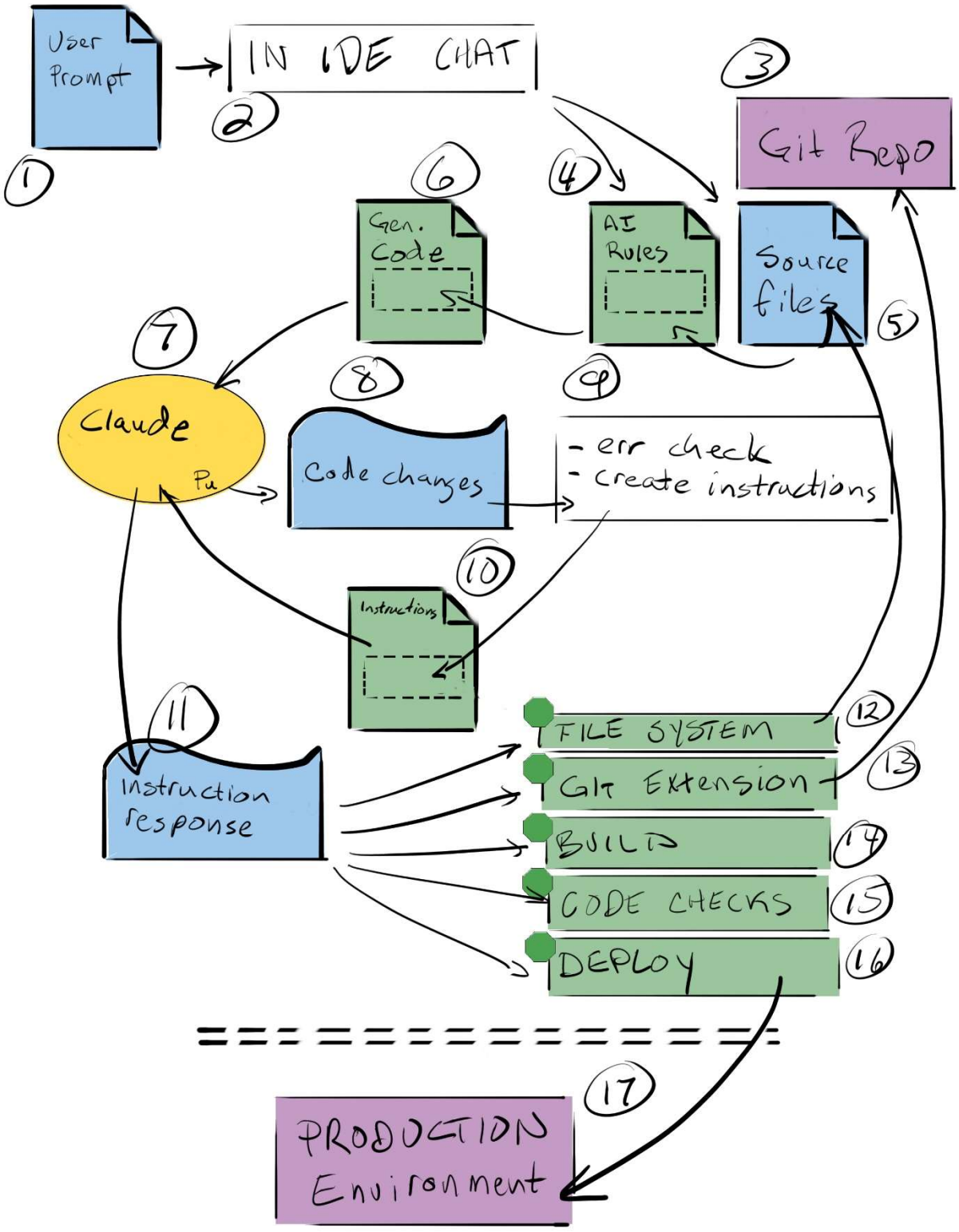

Figure 5 AI Assisted Development Pipeline Example Diagram

AI Assisted User Guide

The following diagram describes an AI based user guide chatbot (2) which takes user queries (1) and looks up document references from a model augmented with training from user guides. The application uses the AI (4) model to transform (3) the user's initial question into a set of keywords (5) and then sends those keywords (6) to the AI to extract a list of references (7) to the different user guides in the company catalog (8). Along the way, system context prompts guide the behavior and format of the AI response. Application code parses and checks the response, preparing it for the next step in the generation process or for presentation (9) to user in the UI (10).

In this case, the diagram indicates where control is achieved via enforcement versus guidance. In places where the AI response (or even user input) might break processing (2, 6, 9, 10) code external to the AI checks if the response seems appropriate to continue. In places where harm is deemed low risk, the system relies on guidance via system context instructions (3, 7) to control AI behavior. The AI component is granted very little power via augmentation, so the team building this system has decided the potential risks are low and possible harms manageable.

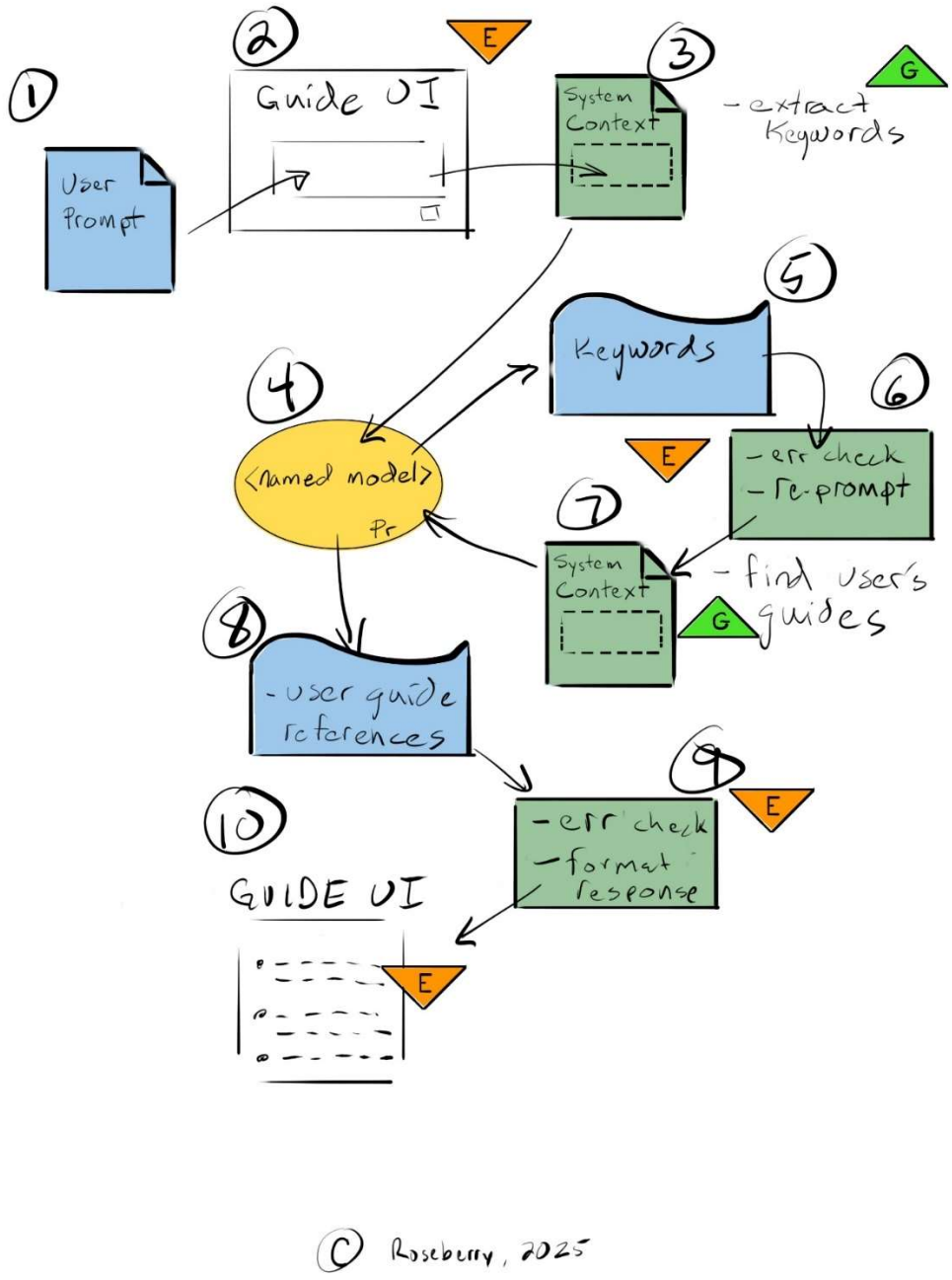

Figure 6 AI Assisted User Guide Example Diagram

Testing Potential Harm, Enforcement Points, Guidance Points (PEG) Based on AIGATE Analysis

There are three main ways AIGATE inspires further testing that break down into the following categories.

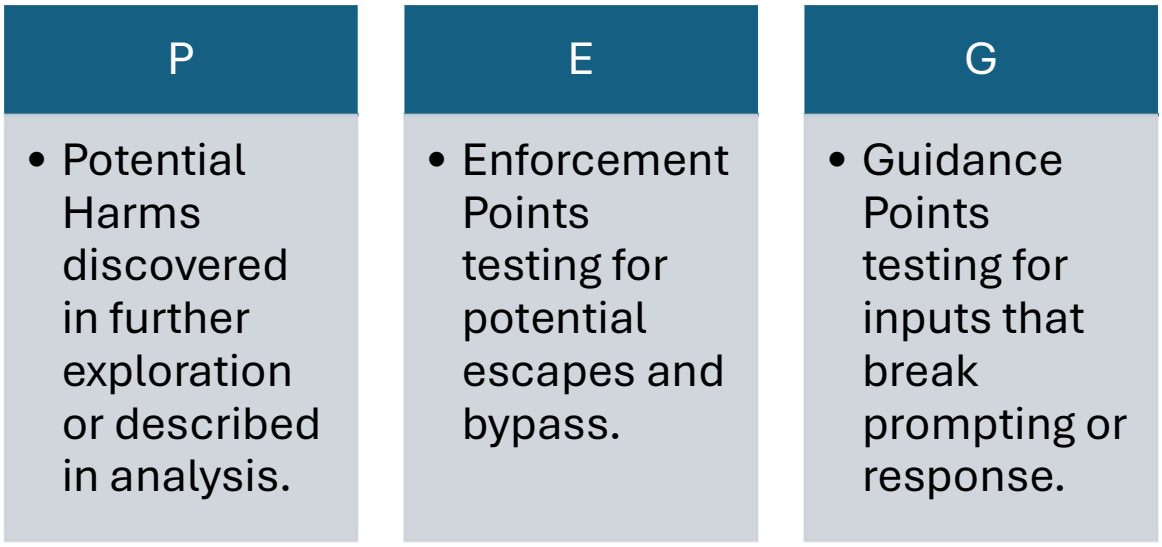

Figure 7 PEG Mnemonic for AIGATE Testing

AIGATE analysis becomes a source of information for testing the safety testing the system. The PEG categories help us think about how to test the system.

Potential Harms

Start with each of the potential harms listed during AIGATE analysis. Consider the following as guiding questions:

- What part of the system protects against those harms, how does it protect?
- What parts of the system introduce the risk of harm and what could induce them to trigger harmful behavior? Pay particular attention to AI augmentation.
- What conditions introduce the risk of harm? How would you create that condition to check whether the protection mechanism fails to prevent the harm?
- How would you detect if the harm was prevented or not?

From that, devious a set of testing methods for exploring those conditions and whether you can discover failures in those mechanisms.

As you continue testing keep the following questions in mind:

- Do any behaviors you observe indicate the possibility of harm not anticipated earlier?
- What changes to the testing approach should we apply to explore those new possibilities?

Enforcement Points

Work from all the enforcement points in the AIGATE analysis. Consider the following questions:

- What are the inputs and outputs at this enforcement point?
- What checks, rules, decisions, or processing happen at this enforcement point?
- What testing permutations of input would exercise each of the checks at the decision point? What inputs might discover defects at the decision point?
- What goes wrong if this enforcement point fails? What might trigger that failure?
- What level of power or control does the AI component (given augmentations) have at this point, and what testing would simulate the range of that power?
- What would well-formed, but problematic, response inputs from the AI look like? What would malformed response inputs from the AI look like?

It is reasonable to simulate AI response input to an enforcement point in order to discover vulnerabilities at that point. It should not be necessary to "jail break" or trick the AI into creating a specific response to demonstrate whether the enforcement point has a defect. This is an extrapolation from the principle "If the AI CAN do something it will eventually do it", but also far more pragmatic than trying to master prompt engineering and AI jailbreak. After discovering the defect, the team can decide whether the bug at the enforcement point merits fixing.

Guidance Points

Take each of the guidance points in the AIGATE analysis. Consider the following questions:

- Where does control of the input to this point come from? Does the system control it, or does something external to the system, such as end users control it?
- What is the nature of the guidance at this point? Are the fragile points in construction of system context or information where variations in the input might disrupt the intended guiding behavior?

- What capabilities and augmentations do the guidance point attempt to affect? What processing takes the AI response as input? Do any of those introduce power and potential for harm not considered before?
- How consistently does the response generated from the prompts guiding behavior match desired outcome? What inputs would exercise the model sufficiently to assess the consistency?

Guidance point testing is about assessing what the model is going to do in response to the context built of the various pieces that comprise the prompt. Sometimes the text in the system context is poorly constructed. Sometimes the inputs from users or external components combine poorly with the system context and cause the AI model to emit undesirable responses that cause problems in the rest of the system. While enforcement point testing involves some degree of simulating the AI model response, guidance point testing will mostly always involve the actual AI model.

Conclusion

The change that AI agents introduce are their capacity to act autonomously with an unpredictable range of behavior. The range of capabilities we give AI agents determines the amount of risk we introduce by building and using them. There are principles that derive directly from the probabilistic nature of all modern AI systems which mean we cannot control directly anything the AI generates. The AIGATE analysis method helps us understand the systems we build and use to benefit as much as we need from AI agents while maintaining control over the behaviors we need as well as those which must not happen. We do this by analyzing the harm and risks we are worried about, understanding which behaviors we must control, and which we tolerate the unpredictability of the AI components. That understood, we build a system with an appropriately selected set of guidance and enforcement points, using that understanding as the basis for testing and hardening the system.

References

Bach, J., & Bolton, M. (2025, October). Testers and AI.