Building Coherent Use into the DevOps Lifecycle for High-Stakes AI

Kelsey Rassmann,[2] Jana Schwartz,[1] Julie Marble,[1] William Regli[1, 2] *

[1] Applied Research Laboratory for Intelligence and Security, University of Maryland, College Park, Maryland
[2] The University of Maryland at College Park
krassman@umd.edu, jschwartz@arlis.umd.edu, jmarble@arlis.umd.edu, regli@umd.edu

Abstract

The creation of high-stakes artificial intelligence (AI) has the potential to positively impact users and their surrounding environment. However, today's development practices introduce the risk of serious errors of misuse, disuse, and abuse that can lead to loss of life and liberty. Software developers must focus on mitigating these potential errors of use while also ensuring the timely delivery of AI-enabled technology. In this paper, we lay out the human-machine interaction in terms of a cost-benefit analysis and thus highlight the need for coherency between the human and AI. We use this reference model to inform development of a Goal, Question, Metric approach to achieving a Coherent Use of AI-enabled software that can be applied throughout the DevOps lifecycle.

Introduction

When developing algorithms and software for artificial intelligence (AI) technology, it is easy to get lost in the technical components, ignoring the intricacies of human interaction. Software developers are self- and team-motivated to meet project goals and maximize technical objectives, but may lose sight of the operational goals of the human-AI system: What is the point of creating incredibly performant software if no one uses it? How can we make sure that operators understand when they should second guess the machine's output rather than follow it blindly into devastating accidents? An algorithm may provide unsurpassed atomic performance, yet not provide the greatest positive impact on mission goals. It is the AI that considers the complexities of human interactions that provides the greatest benefit to the user as well as the surrounding environment.

High-Stakes AI Mustn't "Break Things..." The Silicon Valley "move fast and break things" mentality dramatically predates Facebook, and can be measured by the quantity of software updates that companies like Apple, Microsoft, and Google deploy to their customers. While this practice works well when dealing with low-stakes software (i.e., a phone operating system, targeted advertising, etc.), it becomes problematic as we move towards deploying AI in high-stakes scenarios. In the context of this paper, we describe high-stakes AI as AI that has the potential to cause significant harm to the life or liberty of a user or those impacted by the AI's use. We have already seen this danger with self-driving car technology and facial recognition software. Tesla's Autopilot has led to the death of consumers who have over-relied on their car's ability to drive itself (Krisher 2021), yet Tesla continues to push out beta software with the vague warning "still be careful, but it's getting mature" (Musk 2021; Siddiqui 2021). Loss of liberty is another concern with the advancement of AI. Facial recognition software has already been implemented by some police departments in an attempt to arrest suspects of various crimes (Cooper 2021). However, defects in the software and over-reliance by the users have led to wrongful arrests and jail time for innocent people (General and Sarlin 2021). Belatedly, companies like Amazon have prohibited the police from using their facial recognition software (Weise 2021).

...But We Still Need to "Move Fast" Despite the operational risks, AI still needs to be developed in a timely manner. If too much time is allocated to user testing, by the time the AI is deployed, it is obsolete. This is important in the context of the Department of Defense (DoD). To remain competitive with our adversaries, the DoD needs to act now to start effectively implementing AI (Vergun 2021).

However, the traditional developmental test and evaluation (DT&E) method of software testing is not compatible with the dynamic nature of AI-enabled technology. The DoD has recognized that AI software requires a more flexible testing approach, suggesting the use of software engineering practices like DevOps (or DevSecOps) (Flournoy, Haines, and Chefitz 2020; Software Engineering Institute 2021).

Example: The Z-Virus Diagnosis Decision Aid Consider the following scenario, inspired by movies like (Carnahan, Goddard, and Lindelof 2013), with current high-stakes AI development practices:

The world is experiencing a zombie apocalypse caused by a new virus (Z-Virus). Z-Virus has an incubation period of 5 days before an individual is zombified. If detected early, doctors can cure a Z-Virus infection, but Z-Virus is difficult to detect during the incubation period. A team of developers have come together to create an AI tool to assist doctors in diagnosing patients, the Z-Virus Diagnosis Decision Aid

*Also with the Department of Computer Science

(ZDDA). The team applies a linear, waterfall software engineering methodology, described in (Adenowo and Adenowo 2013), common when creating high-stakes software.

First, the developers identify the requirements of the AI. Then, they use a dataset to train a deep neural network which, given patient data from the doctor, outputs a positive or negative diagnosis in a black-box fashion. Development is followed by extensive model accuracy testing until the ZDDA has achieved an F_1 score of 0.95. Excited by their accomplishments, the development team deploys the ZDDA. However, to their surprise, the incidences of zombified humans only decreases slightly. What happened?

The developers discovered that, in most cases, the doctors using their ZDDA did not trust the output of the AI and chose to diagnose their patients manually. In other failure cases, doctors over-relied on the ZDDA, following its output blindly. The team discovers that the ratio of failures was 5:1 (for every case of coherent use of the AI, there were 5 cases of misuse or disuse). Further, the developers discovered that the ZDDA was only producing an F_1 score of 0.75 post-deployment, as evolutionary mutations caused bias in their training dataset. These development failures caused the ZDDA to have only a slight impact.

Solution: Enabling a Focus on Operational Impact The above example highlights the need to ensure human-AI coherency. This coherency should not be a burden placed upon the user but a responsibility of the developer themselves. Most influences of appropriate reliance and trust in AI correspond to developer decisions that are made as early as design time. However, we cannot expect developers to make the right decisions if we do not provide them the tools to do so while maintaining their development speed.

Thus, in this paper we present the groundwork for a Goal, Question, Metric (GQM) (Basili, Caldiera, and Rombach 1994) approach to achieving a Coherent Use of AI-enabled technology. The goal of our framework is to establish a template GQM breakdown for AI developers based on a model of human-machine interaction. In addition, we provide ideas as to how developers can integrate this approach into the DevOps lifecycle as a means to ensure coherency from the very start of development.

Background

In this section, we explain our reference model of human-machine interaction. Next, we define our concept of "Coherent Use." Finally, we will introduce both the GQM approach and DevOps software development lifecycle, which we apply in the next section to evaluate Coherent Use.

Modeling the Human-Machine Interaction

To understand the importance of coherent AI, it is necessary to model human-machine interaction (HMI). Our model (shown in Figure 1) is based on an idea from (Polizzotto and Molella 2019) that $Value = Benefit/Cost$. We argue, in general, if the benefits outweigh the costs of using a machine ($Value > 1$), the human is likely to use the machine, however, if the benefits do not outweigh the costs ($Value \leq 1$), the human will be unlikely to use the machine. This basic idea is the foundation of our HMI model. We developed this reference model as an abstraction of five spanning use cases; in this paper, we continue using the ZDDA as an example.

HMI Reference Model Overview The model starts when a potentially infected patient goes to their doctor. This *context* from the patient stimulates a *need* for the doctor to help the patient (*GET_NEED*). Then, the doctor determines whether the ZDDA will help them to satisfy their need to help the patient (*NEED_MACHINE_BENEFIT*). If the doctor determines that the ZDDA will not assist them in helping the patient (*NEED_MACHINE_BENEFIT = false*), they will diagnose the patient themselves for a non-Z-Virus ailment (*DO_IT_MYSELF*). Otherwise, if the doctor does believe that the ZDDA is relevant to helping the patient, (*NEED_MACHINE_BENEFIT = true*), they will begin a cost-benefit analysis. Within this analysis, the doctor will determine the *perceived value* of the ZDDA. If it is perceived as low value (*PERCEIVED_VALUE $>$ 1 = false*), then the doctor will diagnose the patient themselves (*DO_IT_MYSELF*). Otherwise, if the doctor believes the *perceived value* of the ZDDA is of high value (*PERCEIVED_VALUE $>$ 1 = true*), then the doctor will have the ZDDA provide them with its output (*PROVIDE_VALUE*). Note that in this first iteration of the HMI model, we skip the *NEED_FULFILLED* action (indicated by * in Figure 1), this is done because *NEED_MACHINE_BENEFIT* covers this condition so an additional check is not needed. From here, the doctor will take the ZDDA's output along with new knowledge regarding the situation (i.e., peer-group doctor gossip, changed patient state, etc., which may each have different update rates) (*NEW_KNOWLEDGE*) and perform another cost-benefit analysis of accepting or rejecting the ZDDA's recommendation (*PERCEIVED_VALUE $>$ 1*). If the doctor perceives the value of the ZDDA as positive (*PERCEIVED_VALUE $>$ 1 = true*), then they accept the diagnosis. This fulfills their need to help the patient (*NEED_FULFILLED = true*) and allows them to deliver this diagnosis to the patient. However, if the doctor perceives the value of the ZDDA as low (*PERCEIVED_VALUE $>$ 1 = false*), then the doctor will reject the diagnosis and diagnose the patient themselves (*DO_IT_MYSELF*). This diagnosis can then be delivered to the patient. Throughout this interaction, both the doctor and the ZDDA deliver diagnosis outcomes to the environment.

HMI Failure Modes The nature of HMI lends itself to a variety of failure modes that are characterized by terminology used within (Parasuraman and Riley 1997): abuse, misuse, and disuse. Abuse refers to a machine that is being used outside the scope of its intended purpose; misuse describes an over-reliance on technology; and disuse is characterized by an under-utilization of technology. As shown in Figure 1, there are two places where these failures happen.

Abuse failures occur when deciding whether or not a machine can fulfill a user's need (*NEED_MACHINE_BENEFIT*). Within the ZDDA example, this occurs if a doctor attempts to use the ZDDA to diagnose a condition other than Z-Virus infection. Misuse and disuse result from an inaccurate perception of value

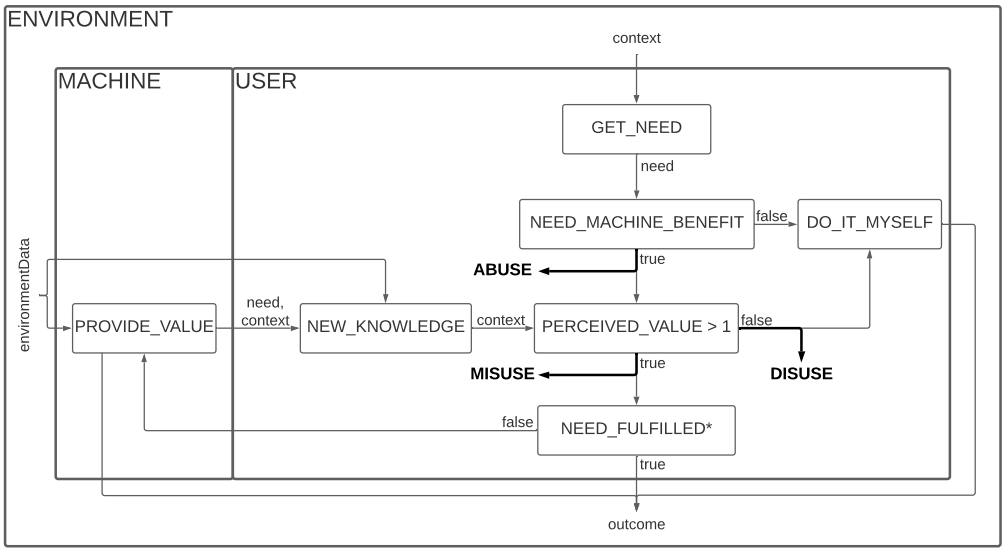


Figure 1: Reference Model of the Human-Machine Interaction.

(*PERCEIVED_VALUE* $>$ *1*). Misuse results if the user perceives the value of the machine as higher than it actually is, thus relying on the machine when they should not (i.e., the doctor accepting an incorrect diagnosis from the ZDDA). Disuse occurs if the user perceives the value of the machine as less than it is (i.e., the doctor rejecting a correct diagnosis from the ZDDA). In this paper, we will focus specifically on the second area of failure (*PERCEIVED_VALUE* $>$ *1*).

Ideally, the user would always make a decision that avoids these error modes (following the non-bold arrows in Figure 1), producing the best outcome on the environment.

Defining HMI Coherent Use

Most developers tend to focus on creating software that provides the best value to the user (focusing on the *PROVIDE_VALUE* function in Figure 1). For instance, a meta-analysis of ICML and NIPS papers show top keywords surrounding "learning," "networks," "optimization," "efficient," etc., with words like "human" and "interaction" not making the list (Kakao AI Report 2017). This focus on performance impacts an AI's trustworthiness: "trustworthy automation is automation that performs efficiently and reliably" (Lee and See 2004). Thus, providing the best possible value to the user is ensuring the trustworthiness of the technology in terms of software performance. However, this emphasis on trustworthiness only represents half of the story. Developers must also focus on the user's perception of value (*PERCEIVED_VALUE* $>$ *1*). This perception of value is termed trust calibration (or simply calibration) in the human-machine trust/interaction literature (Lee and See 2004; Okamura and Yamada 2020; Tomsett et al. 2020). From (Lee and See 2004), "*calibration* refers to the correspondence between a person's trust in the automation and the automation's capabilities (Lee & Moray, 1994; Muir, 1987)." Extending this definition to AI, trust calibration relates to the perceived value of the AI, which is about correctly identifying when to trust the AI and when to question its output.

As figure 2 shows, any gap in trustworthiness (actual value) or trust calibration (perceived value) results in a failure (misuse or disuse). To avoid these errors, we define a combined state of human-AI system use called *Coherent Use*. This state occurs when the machine is trustworthy and allows for trust calibration by the user. This means that the machine provides value to the user and that the user is able to correctly identify its value, thus establishing coherency. We argue that this state is the basis for the best human-machine interaction outcome.

Recent works describing Trustworthy AI (Thiebes, Lins, and Sunyaev 2021; Jain et al. 2020; Varshney 2021), explicitly reference requirements for transparency and explainability. We believe that this community is using the phrase "trustworthy AI" to span both the concept of trustworthiness and that of calibration. We commend their attention and inclusion of calibration features in the popular concept of Trustworthy AI, but assert that trust calibration is important enough when attempting to avoid misuse and disuse, that it should be properly abstracted from trustworthiness. This clarity ensures that trust calibration will get the emphasis that is required, given the impact that it can have on users and the overarching environment.

The Goal, Question, Metric Approach

The Goal, Question, Metric (GQM) software evaluation approach, described by (Basili, Caldiera, and Rombach 1994) utilizes a top-down methodology for identifying the appropriate metrics to achieve high-level goals. The process starts by identifying an overarching goal, then creating quantifiable questions based on that goal, and finally identifying

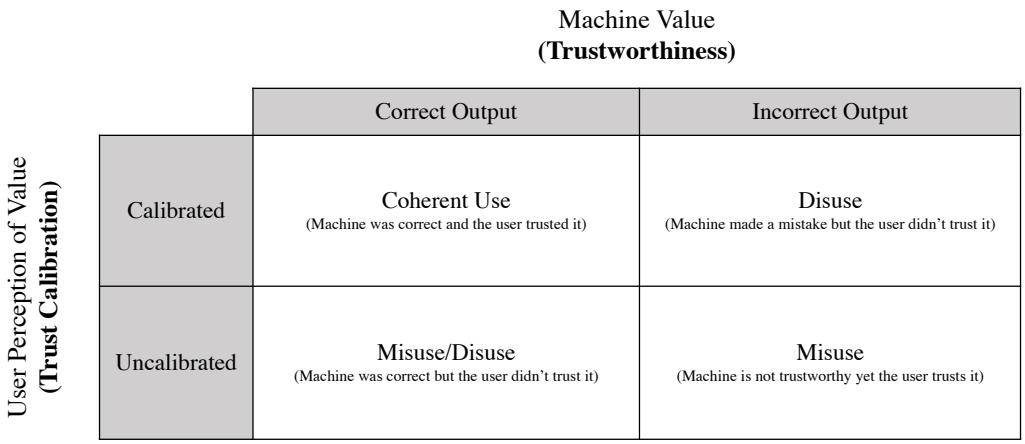


Figure 2: Coherent Use Matrix highlighting various HMI failure modes.

metrics that answer those questions and thus relate back to the goal. By doing this, the paradigm is able to ensure that all measures are characterized by a goal and are not being taken needlessly.

The first step in this approach involves identifying a high-level goal. Goals are made up of four main components: the *purpose* of the measurement, the *object* being measured (taking the form of a product, process, or resource), the *issue*, and the *viewpoint* from which the measurement was taken. Once the goal is established, the questions can be created based off of a model. Given these questions, metrics (both objective and subjective) are determined to answer the questions and measure progress toward the goal.

After the consideration of many different test and evaluation (T&E) techniques, we believe that the GQM approach is the most logical and easy to understand method, thus providing a low barrier to entry for developers who may have never heard of this approach. In addition, the paradigm ensures each measure is taken with purpose which will keep the overall evaluation focused specifically on Coherent Use.

The DevOps Lifecycle

DevOps is a popular software engineering process that seeks to merge the development and operations of a software product. This leads to intercommunication between developers and operators on cross-functional teams that want to deliver continuously at quick speeds (Ebert et al. 2016). The DevOps Lifecycle illustrates how developer practices (plan, develop, verify, test) and operator practices (deploy, operate, and monitor) can be intertwined to create a cross-functional process that integrates operations into the development process. This lifecycle is generally visualized as an infinity with the monitoring phase flowing back into the planning phase. An in-depth explanation of the DevOps lifecycle can be found within (Alnafessah et al. 2021).

Evaluating Coherent Use with GQM

As a means to assert the quick delivery of AI that allows for coherency between the user and the software itself, we propose to integrate our concept of Coherent Use within the DevOps lifecycle through the use of the GQM approach.

Establishing the Goal First, we need to establish the goal of our GQM breakdown. Recall that, a GQM goal is made up of a *purpose*, *object*, *issue*, and *viewpoint*.

Given our discussion of Coherent Use, we will make this concept our main *issue*. Next, we characterize our *purpose* as "maximize", which leads to a goal of maximizing the coherency between the user and the AI. Maximization was chosen specifically because it characterizes the need to minimize occurrences of misuse and disuse, thus allowing the user to avoid the error modes described in Figure 2 as often as possible. The *object* of the goal is the AI-enabled technology whose coherency is being maximized. Since this GQM breakdown is meant as a tool for the developer, the developer will be the one recording metrics. Thus, the *viewpoint* of the goal is that of the developer themselves.

This makes the overarching goal: **Maximize *(purpose)* the Coherent Use *(issue)* of <*the AI-enabled technology*> *(object)* from the viewpoint of the developer *(viewpoint)***.

Creating Questions Since the GQM paradigm bases questions on a characterizing model, we use our reference model of HMI (Figure 1) to establish the components of Coherent Use, namely trustworthiness and trust calibration. We identify four main template questions:

1. Is <*the AI-enabled technology*> trustworthy?
2. Does <*the AI-enabled technology*> allow for trust calibration by the user?
3. Is Coherent Use increasing, decreasing, or remaining constant between delivered software builds? and,
4. Is the current rate of Coherent Use sufficient from the viewpoint of the developer?

Questions 1 and 2 are inherent to our reference model of HMI. These questions describe the components of Coherent Use, specifically, trustworthiness and trust calibration. Question 3 is used as a means to characterize progress toward the goal. However, measuring the maximization of a

property can be difficult, especially since misuse and disuse will likely never be fully mitigated. Thus, we describe the fluctuations in the machine's ability to be coherently used. Finally, Question 4 helps to evaluate the developers perception of sufficiency. Although Questions 3 and 4 are not explicit in the reference model, they are important as a means to not only measure the individual components of Coherent Use (trustworthiness and trust calibration) but also to measure the overall human-AI system performance.

Example Metrics Now that the overarching questions have been established, metrics must be determined that can aid in answering them.

Question 1 is about the trustworthiness of the technology. For the purposes of establishing example metrics for trustworthiness, we continue with the viewpoint that trustworthiness is based on the AI-enabled technology's capabilities. Thus, we can look at trustworthiness in terms of technology performance. More specifically, we view performance as not only raw mathematical performance but also all relevant "-illities" (reliability, robustness, etc.) that pertain to a piece of AI-enabled technology (de Weck, Ross, and Rhodes 2012). Performance will certainly mean different things for different types of AI, which may require developers to come up with metrics that are dependent upon their resources and the intended capabilities of the AI. In general, some example metrics for trustworthiness include overall AI accuracy, robustness, and frequency of failures.

Question 2 discusses trust calibration, which we equate to the perception of the value the AI-enabled technology provides the user. When users are available, and when software builds are mature enough to support user-interactive testing, there are an array of trust and workload measures and metrics. In this case, trust calibration can be measured explicitly, in the field. However, the software development team will require assessment of this question before the technology is mature, and more frequently than users may be leveraged. Within the literature, trust calibration is discussed in terms of transparency and explainability (Tomsett et al. 2020; Boyce et al. 2015; Okamura and Yamada 2020). Thus, we focus our trust calibration metrics (when a user is not available) on the transparency of the software. This includes not only transparency as the AI delivers an output to the user but also transparency in terms of making users aware of the limitations of the AI-enabled technology's use. Some example metrics for trust calibration include the number of misuse and disuse accidents, the performance/correctness of an explainable AI interface, and communicated limits of use.

Question 3 focuses on ensuring progress toward the goal, specifically measuring the fluctuations of Coherent Use. This can be done by analyzing the overall human-AI system performance. Measurements include the frequency with which the machine is used compared to past frequency metrics, and the current number of misuse and disuse accidents compared to the past number of accidents.

Finally, Question 4 is characterized by the developer's perception of Coherent Use. This question can be answered using subjective analysis by the developer given their understanding of the coherency of the software product.

GQM Integration with DevOps

AI must be developed and deployed with speed and agility. Thus, we adapt the previous GQM template to fit within the DevOps lifecycle.

During development, there are times when certain capabilities or the entire software product itself does not exist yet to be tested. This creates the need to be able to assess trustworthiness and trust calibration when a piece of software is at different phases of development. Thus, we break down the DevOps lifecycle into three main GQM testing phases:

1. Active Development (Plan, Develop);
2. Pre-Deployment Testing (Verify, Test, Deploy); and,
3. Post-Deployment Testing (Operate, Monitor).

Each of these phases has its own set of questions and associated metrics. During the first iteration of the Active Development phase, the AI-enabled technology does not yet exist and therefore the questions are focused on a *potential* for trustworthiness and trust calibration, and questions are answered in terms of design considerations and software requirements analyses. On the second iteration of this phase, the developer perception of AI coherency can also be measured. In the Pre-Deployment Testing phase, the questions focus on the *performance* of the AI, including the performance of transparency methods meant to aid in trust calibration. Finally, during the Post-Deployment Testing phase, questions are formed in terms of how the AI is performing in the real world regarding trustworthiness and the ability for users to calibrate their trust. This phase is also where the question of overall human-AI coherency emerges, as there is now a full human-AI system where user interactions can be analyzed. An overview of these questions and metrics can be seen in Figure 3, note, that questions and metrics with a * are used after the first DevOps cycle.

Using the GQM Template

Given the template described above, the AI development team can use the goal template by substituting their AI-enabled technology as the *object* of the goal. From there, the questions at the corresponding phases of the DevOps lifecycle can be used as a means to guide the selection of appropriate metrics. Depending upon the type of AI and what can feasibly be measured by the developer, metrics are determined that make the most sense for the software product.

Predicted Impact

In terms of cost-savings, the IBM System Science Institute performed a cost-savings analysis based on when security bugs were found during a traditional software development approach (waterfall model). They found that bugs discovered during testing were 15x more expensive to fix than bugs found during the design phase (Dawson et al. 2010). While these metrics focus on security, we believe they extend to bugs in Coherent Use as they still require immediate remedy for the safety of the user, especially when utilizing AI in high-stakes scenarios.

GQM Testing Phase	DevOps Phase	Questions	Example Metrics
Active Development	Plan	**Q1.1:** Does *<the AI-enabled technology>* design promote trustworthiness?	**M1.1:** AI accuracy requirements; AI robustness requirements
	Develop	**Q1.2:** Does the *<the AI-enabled technology>* design create a potential for trust calibration?	**M1.2:** # of requirements asserting AI transparency; design meeting notes
		Q1.3: Is the current rate of Coherent Use sufficient from the viewpoint of the developer?*	**M1.3:** Developer subjective understanding of current AI coherency*
Pre-Deployment Testing	Verify	**Q2.1:** Is *<the AI-enabled technology>* showing trustworthy behavior in controlled scenarios?	**M2.1:** AI Accuracy; AI Reproducibility; # Passed Requirements
	Test	**Q2.2:** Are *<the AI-enabled technology>*'s components related to trust calibration working properly?	**M2.2:** Accuracy of XAI/Model Confidence; Documentation of AI Performance; Documentation of Limits of Use
	Deploy		
Post-Deployment Testing	Operate	**Q3.1:** Is *<the AI-enabled technology>* showing trustworthy behavior post-deployment?	**M3.1:** Post-Deployment Accuracy; Incidences of Software Failure
	Monitor	**Q3.2:** Are users able to calibrate their trust while using *<the AI-enabled technology>?*	**M3.2:** # Misuse/Disuse Accidents
		Q3.3: Are instances of Coherent Use increasing, decreasing, or remaining constant between delivered software builds?	**M3.3:** Compared # of Misuse/Disuse Accidents; Compared Software Usage

Figure 3: Varying questions and example metrics over the DevOps lifecycle.

ZDDA Example

To demonstrate our proposed GQM approach, we return to the ZDDA example. Instead of using the traditional T&E method described previously, this time the team uses our GQM approach to ensuring Coherent Use. To do this, they first establish their goal as: **Maximize *(purpose)* the Coherent Use *(issue)* of the ZDDA *(object)* from the viewpoint of the developer *(viewpoint)***. Next, they begin development while establishing questions and metrics for each GQM testing phase using the template laid out in Figure 3.

Phase 1: Active Development While actively developing the ZDDA, the developers establish the following questions during the first iteration:

- **Q1.1**: Does the ZDDA design promote trustworthiness?
- **Q1.2**: Does the ZDDA design create a potential for trust calibration?

After the initial planning and development phases, the developers answer these questions using a variety of metrics. For example, a metric that could be used for Q1.1 is the diagnosis F_1 score asserted by the software requirements. For the sake of this example, let's say this value is 0.90. Q1.2 could be answered with a metric describing the number of requirements asserting algorithm transparency, let's say that after initial planning this is 0 requirements (recall that the developers were originally creating a black box algorithm). While the accuracy score intended for the AI is seemingly sufficient, the developers quickly realize by using this framework that they need to create more requirements related to the transparency of their algorithm. Thus, they decide to add a feature that displays the confidence level of the AI with its diagnosis, which creates 5 new software requirements. The developers decide that this is satisfactory to achieve a potential for both trustworthiness and trust calibration and therefore move onto the next phase.

Phase 2: Pre-Deployment Testing The following questions are created for the Pre-Deployment Testing phase:

- **Q2.1**: Is the ZDDA showing trustworthy behavior in controlled scenarios?
- **Q2.2**: Are the ZDDA's components related to trust calibration working properly?

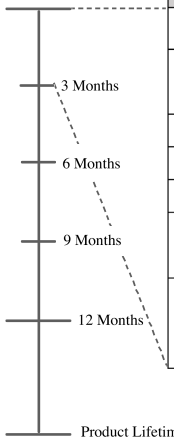
The developers answer these questions with an F_1 score of the ZDDA algorithm (Q2.1) and a model confidence accuracy percentage (Q2.2).

The accuracy percentage for model confidence represents the percentage of the time the model confidence output is within 5% of the true model confidence percentage. In this example, the developers found that their algorithm achieved an F_1 score of 0.9 while their model confidence feature achieved only 60% accuracy. From these results, the developers were able to iterate on their model confidence algorithm and build the accuracy to 95%. These metrics pass the requirements established in the design phase and allow the developers to deploy the ZDDA.

Phase 3: Post-Deployment Testing The questions created in the Post-Deployment Testing phase consist of:

- **Q3.1**: Is the ZDDA showing trustworthy behavior post-deployment?
- **Q3.2**: Are users able to calibrate their trust while using the ZDDA?
- **Q3.3**: Are instances of Coherent Use increasing, decreasing, or remaining constant between delivered software builds?

Traditional Waterfall Method (1 Year of Development)	
Waterfall Phase	Summary
Requirement Analysis	No bugs found
Design	No bugs found
Implementation	Minor bugs found and fixed throughout development
Testing	Performance fixes until $F_1 = 0.95$
Operation and Maintenance	**Bug found –** High levels of misuse and disuse (5:1 ratio) *__Defect costly to fix__*



GQM for Coherent Use w/ DevOps Integration (Quarterly Releases – 3 Months of Development)		
DevOps Phase	GQM Testing Phase	Summary
Plan	Active Development	**Bug Found –** Lack of system transparency requirements
Develop		
Verify	Pre-Deployment Testing	**Bug found –** Inaccurate model confidence
Test		
Deploy		
Operate	Post-Deployment Testing	**Bug found –** Inaccuracy of model **Bug found –** Elevated levels of misuse and disuse (1:10 ratio)
Monitor		

Figure 4: Comparison of the original development of the ZDDA using Waterfall phases, described in (Adenowo and Adenowo 2013), and the development of the ZDDA using the GQM breakdown for Coherent Use throughout the DevOps lifecycle.

These questions are answered by measuring the post-deployment overall ZDDA F_1 score (Q3.1) and the ratio of misuse and disuse incidences compared to incidences of Coherent Use (Q3.2 and Q3.3). The developers find that their AI only achieves an F_1 score of 0.75 post-deployment and the misuse and disuse ratio is 1:10 (for every case of misuse/disuse there are ten cases of Coherent Use).

Starting with Q3.1, the developers discover the decreased accuracy is due to bias in their training data set. The misuse and disuse ratio shows that while doctors are able to calibrate their trust, there is still more that the developers could be doing to help this process, answering Q3.2. At this stage Q3.3 cannot be answered since the level of Coherent Use has not been measured previously, however, we include this question in this first DevOps iteration because it establishes the importance of measuring Coherent Use during this phase. This metric can be used in the subsequent iterations of the DevOps lifecycle to answer this question.

Phase 1: Active Development, Again After going through the DevOps cycle once, the information gathered from the earlier iteration can be used to inform new requirements and feature development. Within this phase, the two previous questions are asked again, however, this time a third question can be asked as follows:

- **Q1.3**: Is the current rate of Coherent Use sufficient from the viewpoint of the developer?

This additional question helps to drive the developers toward continually improving their software. In this case, while the AI performed well, the developers were not satisfied with the rates of Coherent Use that were measured during Post-Deployment Testing. Thus, the developers decide to add requirements asserting the scope of their training data set in order to avoid errors of inaccuracy post-deployment. They also add additional transparency feature requirements (i.e. an explainable AI interface), to help aid in reducing the cases of misuse and disuse even further.

Discussion

The Z-Virus example highlights why this framework is needed in practice. It opens the door for conversation surrounding Coherent Use from the very start of development, thus allowing for cost savings and an operationally impactful deployed product. The developers were able to realize, within their DevOps lifecycle, where the ZDDA was going to fall short and fix those problems preemptively rather than wait until the end of the lifecycle. This was seen when the developers added an additional feature during the Active Development phase, fixed a flaw in their model confidence feature during the Pre-Deployment Testing phase, discovered problems with their training data set during the Post-Deployment Testing phase, and added even more requirements during the second iteration of the Active Development phase. Assuming quarterly releases (every 3 months) using the GQM DevOps approach and one year of development for the traditional approach, Figure 4 shows a comparison of these two methodologies. The GQM T&E methodology not only deployed after only 3 months of development, but also allowed for increased iteration on the software. This provided significant cost savings compared to the traditional approach, which did not discover major errors of misuse and disuse until after deployment. While the product created by the GQM process was not perfect, this approach made the first deployment of the ZDDA more effective at stopping the zombie apocalypse and informed the developers on additional features that could improve the product even further.

Conclusions and Future Work

Our whimsical example demonstrates that the current AI development practice does not suffice when creating high-stakes AI, where errors of misuse and disuse can lead to devastating losses of life and liberty. It is the responsibility of the development team to ensure that they produce high-quality (trustworthy) AI that allows the user to accurately perceive this quality (trust calibration); without this consideration, there is a high potential for misuse and disuse of the software. From our reference model of HMI, we proposed a GQM approach that seeks to maximize the Coherent Use of an AI. This approach is embedded within the DevOps lifecycle to help maintain development agility.

Future work on this GQM template involves maturing the language used for each of the template questions as well as developing a deeper understanding of the correct metrics to use in certain product scenarios. In addition, the template can be tested on an actual (rather than hypothetical) AI where real data can be gathered regarding trustworthiness and trust calibration.

References

Adenowo, A. A.; and Adenowo, B. A. 2013. Software Engineering Methodologies: A Review of the Waterfall Model and Object-Oriented Approach. *International Journal of Scientific & Engineering Research*, 4(7): 427–434.

Alnafessah, A.; Gias, A. U.; Wang, R.; Zhu, L.; Casale, G.; and Filieri, A. 2021. Quality-Aware DevOps Research: Where Do We Stand? *IEEE Access*, 9: 44476–44489.

Basili, V. R.; Caldiera, G.; and Rombach, H. D. 1994. The Goal Question Metric Paradigm. *Encyclopedia of Software Engineering*, 528–532.

Boyce, M. W.; Chen, J. Y.; Selkowitz, A. R.; and Lakhmani, S. G. 2015. Effects of Agent Transparency on Operator Trust. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts*, HRI'15 Extended Abstracts, 179–180. New York, NY, USA: Association for Computing Machinery.

Carnahan, M. M.; Goddard, D.; and Lindelof, D. 2013. World War Z. Screenplay.

Cooper, A. 2021. Police departments adopting facial recognition tech amid allegations of wrongful arrests. https://www.cbsnews.com/news/facial-recognition-60-minutes-2021-05-16/. Accessed: 2021-10-26.

Dawson, M.; Burrell, D. N.; Rahim, E.; and Brewster, S. 2010. Integrating Software Assurance into the Software Development Life Cycle (SDLC). *Journal of Information Systems Technology and Planning*, 3: 49–53.

de Weck, O. L.; Ross, A. M.; and Rhodes, D. H. 2012. Investigating relationships and semantic sets amongst system lifecycle properties (ilities).

Ebert, C.; Gallardo, G.; Hernantes, J.; and Serrano, N. 2016. DevOps. *IEEE Software*, 33(3): 94–100.

Flournoy, M. A.; Haines, A.; and Chefitz, G. 2020. Building Trust through Testing. https://cset.georgetown.edu/event/building-trust-through-testing/. Accessed: 2021-12-11.

General, J.; and Sarlin, J. 2021. A false facial recognition match sent this innocent Black man to jail. *CNN Business*.

Jain, S.; Luthra, M.; Sharma, S.; and Fatima, M. 2020. Trustworthiness of Artificial Intelligence. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 907–912.

Kakao AI Report. 2017. Meta-analysis on 6,163 papers of ICML&NIPS. https://medium.com/@kakaoaireport/meta-analysis-on-6-163-papers-of-icml-nips-cbef530eaaf6. Accessed: 2021-12-09.

Krisher, T. 2021. 3 crashes, 3 deaths raise questions about Tesla's Autopilot. *AP NEWS*.

Lee, J. D.; and See, K. A. 2004. Trust in Automation: Designing for Appropriate Reliance. *Human Factors*, 46(1): 50–80. PMID: 15151155.

Musk, E. 2021. If you want the Tesla Full Self-Driving Beta downloaded to your car, let us know. Doubling beta program size now with 8.2 & probably 10X size with 8.3. Still be careful, but it's getting mature.

Okamura, K.; and Yamada, S. 2020. Adaptive trust calibration for human-AI collaboration. *PLOS ONE*, 15(2): 1–20.

Parasuraman, R.; and Riley, V. 1997. Humans and Automation: Use, Misuse, Disuse, Abuse. *Human Factors*, 39(2): 230–253.

Polizzotto, L.; and Molella, A. 2019. The Value Balance. *IEEE Engineering Management Review*, 47(4): 24–31.

Siddiqui, F. 2021. Tesla owners can now request 'Full Self-Driving,' prompting criticism from regulators and safety advocates. *Washington Post*.

Software Engineering Institute. 2021. 2020 SEI Year in Review. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Thiebes, S.; Lins, S.; and Sunyaev, A. 2021. Trustworthy artificial intelligence. *Electronic Markets*, 31(2): 447–464.

Tomsett, R.; Preece, A.; Braines, D.; Cerutti, F.; Chakraborty, S.; Srivastava, M.; Pearson, G.; and Kaplan, L. 2020. Rapid Trust Calibration through Interpretable and Uncertainty-Aware AI. *Patterns*, 1(4): 100049.

Varshney, K. 2021. Foundations of trustworthy AI: How to conduct trustworthy AI assessment and mitigation. https://www.ibm.com/blogs/watson/2021/06/trustworthy-ai-assessment-mitigation/. Accessed: 2021-12-11.

Vergun, D. 2021. Artificial Intelligence Key to Maintaining Military, Economic Advantages, Leaders Say. https://www.defense.gov/News/News-Stories/Article/Article/2567486/artificial-intelligence-key-to-maintaining-military-economic-advantages-leaders/. Accessed: 2021-12-11.

Weise, K. 2021. Amazon indefinitely extends a moratorium on the police use of its facial recognition software. *The New York Times*.