The CDD Playbook

RASA

Table of Contents

Introduction

Conversational interfaces are open ended—a source of their appeal to users and a challenge to the teams who build them. How can you anticipate what users might say when they could say...anything?

The truth is, trying to anticipate every conversation path or message is impractical or even impossible. Development teams can spend many cycles building out conversation paths they expect users to take, only to find users are inclined to take another path entirely. Successful teams avoid building speculative features and instead take their cues from users.

AI assistants offer a built-in solution to the challenge of learning what users want. The best predictor of what future users will say to your assistant is what users have said in the past. We can use conversation data to understand the user based on past interactions and even feed conversations back into the assistant as training data, allowing the assistant to learn and better recognize what users are saying over time. Users tell us what they want through the assistant, in their own words.

This principle is one key idea behind conversation-driven development (CDD): that we can build better AI assistants when we listen to users and use what they say to guide our development. At Rasa, we came together around CDD because we realized that a conversational AI framework alone wasn't enough to build Level 3 assistants and beyond. Technology will get you part of the way, but design and development practices also play a large role. From your team structure to your workflows, the methodology around building AI assistants is just as important as the tools.

Conversation-driven development doesn't stop with leveraging conversation data; it also encompasses the engineering practices that help teams build resilient, reliable assistants. CDD touches every part of the development process, from design, to testing, to DevOps. This set of principles captures what we've learned by working with thousands of chatbot developers, lessons that provide a blueprint for building better AI assistants.

Conversation-Driven Development

CDD isn't proprietary, and it isn't tied to a particular framework. It's a set of activities and design principles that help conversational AI teams build AI assistants that really help users.

Conversational interfaces give developers unique insight into user feedback because every conversation users have with an AI assistant is recorded in the assistant's database. Instead of making educated guesses about how well users' needs are being met, you can see exactly how each interaction went in the conversation data. Even better, these conversations can become training data that helps your model make better and more accurate decisions over time.

One of the core ideas of CDD is making full use of conversation data: conversations provide training data for your model and guide your development decisions by showing you what is and isn't working about your assistant's design. The other piece of CDD is rooted in engineering best practices and DevOps.

We've incorporated these ideas into six steps that make up CDD:

- *Share* - Test your prototype early in the development process with users from outside your development team.
- *Review* - Read the conversations that users have with your assistant, instead of focusing entirely on top-level metrics.
- *Annotate* - Convert the messages that users send to your assistant into training examples.
- *Test* - Make testing an automated step, every time you deploy new changes.
- *Track* - Measure meaningful metrics, so you can understand what's working and not working.
- *Fix* - Reduce failures over time by making adjustments based on analyzing your assistant's interactions.

Teams building an AI assistant will likely perform several of these steps simultaneously and move back and forth between them. The process also involves multiple roles and skill sets. Developers, content creators, and product owners all work together to make CDD happen.

Rasa X

While CDD isn't specific to any single technology, it's easier to do with the right tooling. We developed Rasa X to be the tool that would make it simple to sift through conversation data and make it actionable. Rasa X layers on top of Rasa Open Source and provides a UI for reviewing conversation data and annotating user messages. It also includes features for testing your assistant and sharing it with other users before you go live.

We'll show you how Rasa X can be used to support CDD throughout this playbook.

CDD at Rasa - How we know it works

The idea of conversation-driven development was born out of our work with customers and through building assistants of our own. Qualitative feedback from users showed us that we were on the right path, but we wanted to understand the tangible benefits of building assistants using these methods.

We recently conducted an experiment to compare CDD methodology with techniques to generate training data that are contrary to CDD: for example, using a script to automatically generate a large number of paraphrased training phrases. We compared two assistants side-by-side, identical except for their NLU training data. Our CDD assistant was shared with a large group of testers early in development, and the user messages collected from test conversations were used to build up the assistant's NLU training data. The other assistant's training data was automatically generated by a script and supplemented with phrases written by the development team.

We then tested the accuracy of each assistant's NLU model. As we'd imagined, the CDD model was able to generalize better than the model trained on hypothetical and autogenerated data. The autogenerated data set was larger, but the CDD dataset was representative; that is, it contained the types of messages real users are likely to send. A 5-fold cross-validation on a model trained on the CDD data produced an F1 score of 90%, whereas a model trained on the autogenerated data and tested against the CDD data set scored just 81%.

If you're thinking it sounds like the CDD data set has an unfair advantage, you're right, but not for the reasons you might think. It's true that the CDD model was trained and tested from the same distribution, but this isn't too far off from what the model would encounter in the real world. As you practice CDD over time, you'll reach a point where most of the messages users send will already be in the training

data—and DIET (our NLU architecture) typically fits to training data messages with 100% accuracy.

When we tested the autogenerated model against its own autogenerated data, the F1 jumped to 94%, but keep in mind that this only tests the model's ability to classify the same type of manufactured messages it was trained on. If we test the same model against real-world messages from the CDD data set (a much better indicator of real-world performance), the F1 score is a full 13 points lower.

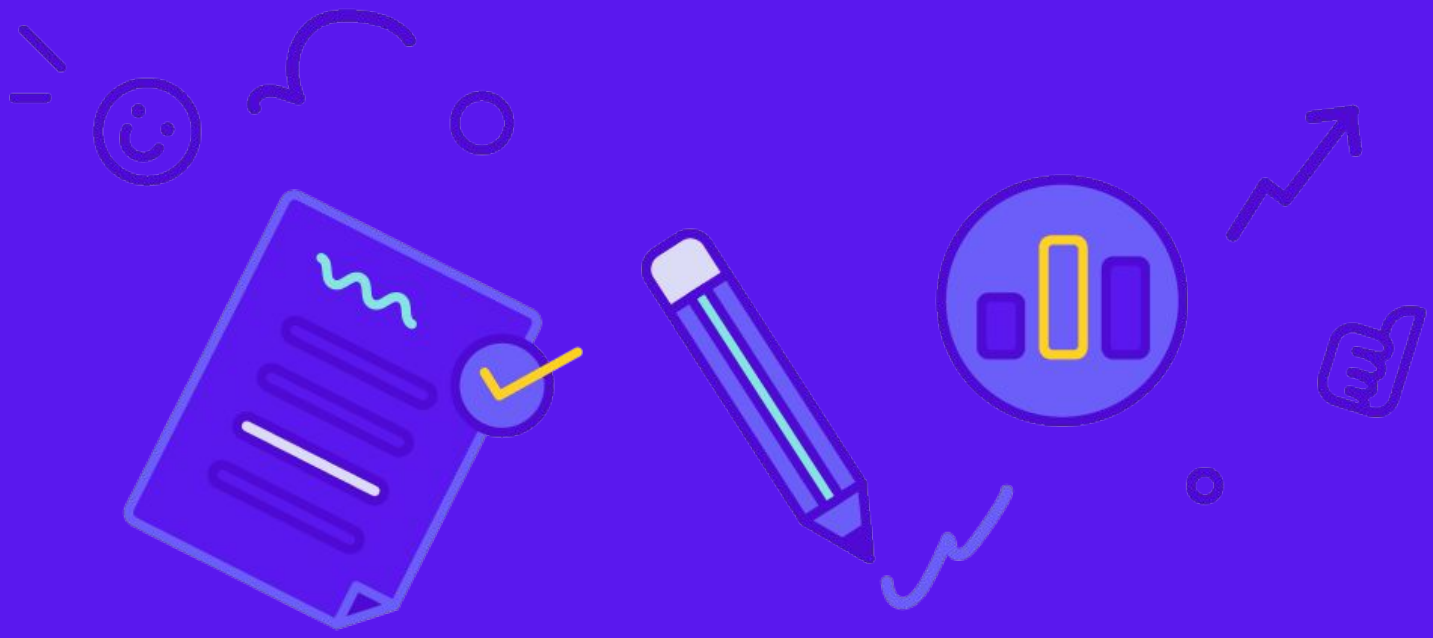
But there's more to CDD than creating data from real-world user messages. The six steps of CDD are drawn from our experience building assistants at Rasa. Whether it's tracking success rates in Carbon bot, an assistant built by the Rasa research team, or the CI/CD workflow we use to release updates to our starter packs and demo assistant, these methods have been tried and tested by the Rasa team, and popularized by the community.

How to Use this Playbook

This playbook includes 5 guided activities to help you put conversation-driven development into practice. The plays are designed for developers, product owners, and conversation designers—anyone who builds AI assistants and wants to create a better experience for their users.

Within each play, you'll find deep dives on the concepts behind CDD, as well as activities to do with your team and questions to spark discussion. At the end of the playbook, you'll find a checklist for tracking your progress and additional resources.

To get the most value out of the plays, you'll need to have an AI assistant that's at least in the prototype phase. If you haven't started building your assistant yet, you can still read along to get familiar with the principles behind CDD, but you'll need a simple assistant to follow along with the hands-on activities. Check out a few of the resources here if you're just getting started building your assistant. You can also use one of the Rasa Starter Packs as a pre-built starting point.



CDD Self-Assessment

CDD Self-Assessment

Before you can begin to align on a conversation-driven development as a team, it's important to measure where your team stands today. We've created a scoring matrix to help you benchmark your team's current practices and identify opportunities.

This play is designed to help your team come together on what CDD means and take a closer look at the habits and practices that are currently in place. This play can also help you prioritize which practices to work on and create a CDD roadmap for your team.

Play 1: Self Assessment

Step 1: Discuss objectives

As the facilitator, set the stage for the discussion by providing background on conversation-driven development and outlining the goals for the session. Tie the discussion back to what your team hopes to accomplish with your assistant; for example, increasing your assistant's accuracy or lowering the abandonment rate, or reducing incidents associated with deployment.

Materials:

- CDD Scorecard Template (pg. 11)

Time:

- 60 minutes

People:

- 1-10 participants
- 1 facilitator

Step 2: Complete self-evaluations as a team

Give each team member a copy of the CDD Scorecard Template and allow 15-20 minutes for all participants to score the team. For each category (Conversation review, Training data, User testing, DevOps, Metrics), 1 point should be awarded for Stage 1, 2 for Stage 2, and so on. Participants should record their score for each category, total the category scores, and then divide by 5 to reach an average. Your average score corresponds to your overall CDD Stage.

Step 3: Share the self-evaluations

Each participant reports their scores to the group and shares first impressions. Pay close attention to categories where scores varied widely among participants and discuss why opinions differed.

Step 4: Determine next steps

Identify 1-2 areas where the team can improve their score. Brainstorm quick wins and pick one idea the team can put in place in the short term.

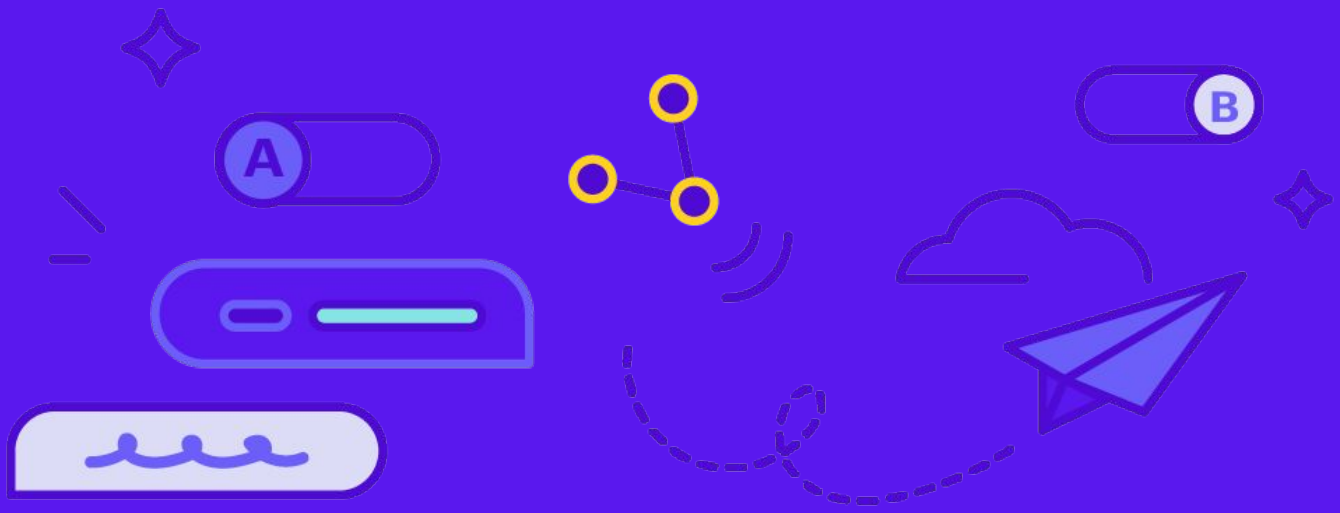
CDD Scorecard

	Stage 1	Stage 2	Stage 3	Stage 4	Score
Conversation review	Conversation data is not accessible. Team lacks the technical capability to access historical conversation data.	Conversation data is collected but not leveraged. Conversations are not read or reviewed, and no top-level metrics are calculated.	Top-level metrics, like contact volume and most-used intents are used exclusively to measure the assistant's success rate. Conversations are reviewed in a spreadsheet or other static tool where actions are taken separately from analysis.	Conversations are analyzed to produce top-level metrics and are read individually by team members. Metrics and qualitative insights are used by the team to guide development and business decisions. Conversations are reviewed in a tool where team members can analyze and complete follow-up tasks at the same time.	
Training data	Training data is written by the development team or generated using paraphrasing tools.	Training data is written by conversation designers and SMEs based on historical chat transcripts, search terms, and FAQs.	Training data written by team members is supplemented with annotated messages from conversations between the assistant and real users. About 30% of data comes from annotated user messages.	Annotated user messages make up 90% of training data. Scope and tasks handled by the assistant are heavily influenced by conversation data.	
User Testing	The development team does not conduct usability tests with users. Primary channel for user feedback is the customer service team.	Usability tests are conducted only after launch. The assistant is not given to users outside of the development team until it has been deemed production-ready	The development team conducts a beta test when the assistant is ~80% complete. Main purpose of beta testing is to identify remaining blockers prior to launch.	The assistant is given to users as soon as a working prototype is ready and continues to test throughout the development process. Conversation data and feedback is quickly incorporated, and the development team works in short, iterative cycles.	
Deployment workflow	After making changes, the developer manually tests the assistant by talking to it. Code updates are pushed directly to production without review.	Pull requests must be approved by another team member, who tests the changes manually on their computer. Updates are pushed directly to production after approval.	Updates are pushed to a separate staging environment for manual testing before they are released to production. A basic CI/CD pipeline is in place, but does not include automated testing.	CI/CD pipeline includes automated unit and integration tests as well as automated model testing. PRs are reviewed by team members before merging, and functional tests are conducted in a separate staging environment before release to production.	
Metrics	No metrics are tracked to determine the assistan't success rate. Success is measured based on feedback from internal stakeholders.	Basic metrics like chat volume are tracked. Main success metric is number of contacts handled by the assistant.	Top level metrics like volume, intent frequency, and NPS are used to measure the assistant's success rate. Metrics seek to measure both frequency and quality of contacts.	Top level metrics are measured in combination with qualitative measures based on conversation review. External signals, like whether a user re-contacted support within 24 hours, are also considered.	

For each category (Conversation review, Training data, User testing, DevOps, Metrics), award 1 point for Stage 1, 2 for Stage 2, and so on. Total all category scores and divide by 5 to calculate the average. Your average score corresponds to your overall CDD Stage.

Discussion Questions

1. How does the team decide which new intents and training examples should be added?
2. Which success criteria is the assistant evaluated against?
3. How often do deployments result in unexpected behavior or bugs?
4. Are there any pain points around coming up with new training examples?
5. How much access do team members have to first-hand user research?
6. What is the current process for reviewing conversation data? Which tools are used to sort and prioritize conversations?
7. How much context-switching is required when reviewing conversations and taking action based on what you've found?



Share your Assistant

Share Your Assistant

Bringing users into the development process—early and often—is one of the cornerstones of conversation-driven development. We typically think of user testing as a method to validate whether or not the user's problem is being solved, and this is certainly a good reason to get your assistant into the hands of users. A user test can provide incredibly valuable insight into users' expectations and behaviors that can't be uncovered in a simple interview. And understanding this feedback early on allows you to course correct and modify your design before it becomes too costly or technically complex.

But, there's another good reason to test your assistant with users while you're still in the early stages of development: it helps you build a representative data set. User testing doesn't just measure how effective your design is; the process also generates messages, which can be annotated and turned into training data, helping you build your assistant at the same time.

If possible, you should recruit testers who reflect the demographics of your end user, but early tests can also be conducted using volunteers from your organization. The most important thing is that testers should **not** have first-hand knowledge of the assistant's development. Testers who were involved with building or designing the assistant tend to subconsciously stick to the messages and conversation flows they know the assistant can handle. Even though it might be painful to see your assistant fail, test users who stretch the limits of your assistant's capabilities will give you the best insight into how to prepare your assistant for the real world.

*The most important thing is that testers should **not** have first-hand knowledge of the assistant's development.*

We recommend running this play as soon as your assistant can handle a few basic conversation paths, and we also recommend using Rasa X to share your assistant with test users while it's still in the early stages. Rasa X provides a simple chat interface, so you can test before you connect a messaging channel, and your tester's conversations are automatically saved for you to analyze. You can run this play as often as you need to, while your assistant is in development, and later, after you release your assistant to production.

Play 2: Conduct a User Test

Materials:

- Rasa X instance, running locally or on a server
- Laptop or mobile phone for each tester

Time:

- 2-3 hours

People:

- 3-10 testers
- 1 note taker
- 1 facilitator

Step 1: Prep for the user test

Before you begin, decide on a list of tasks you want the tester to complete. Provide enough contextual information for the tester to complete the tasks without leading the tester or influencing their behavior. Prepare a list of 10-15 questions you'd like to ask the user during the session.

Designate a note taker from your team to record observations during the session. This allows the facilitator to concentrate on asking questions and guiding the session, and it brings other team members into the process as active participants.

Step 2: Explain the process

At the beginning of the session, warm up your testers by providing background information about your project and outlining what you'll be doing during the test session. Encourage your testers to open up and be candid—it's important to convey that you're testing the assistant, not them. If you're conducting your user test remotely, this step can be accomplished via email.

Step 3: Share a test link

If you're running Rasa X locally, you'll need to run ngrok to make your assistant available to your testers on their own computers. If you're running Rasa X in server mode, the test link will be publicly accessible by default.

Generate a Share your bot link using Rasa X and distribute the link to each tester. If your users are on mobile devices, we recommend converting the link to a QR code.

Ask your testers to complete the tasks you've outlined and encourage them to describe their thought process out loud, especially if they find something frustrating or surprising.

Step 4: Interview the testers

Interview your testers using the questions you prepared. Let your testers do most of the talking—you can follow up with statements such as, "and then what happened?" to encourage testers to keep going and open up.

Step 5: Discuss and document

As soon as the session has ended, reflect on what you've learned as a team while the session is still top of mind. Then, review the conversations you've collected. Check out Play 3 for more tips on reviewing and annotating conversations.

Discuss important feedback and publish the notes and summary of the test session in a place where they're accessible to others, both on your team and across the organization. Create a list of next steps for turning what you've learned into improvements for your assistant.

Discussion Questions

1. Were users able to return to the happy path if they got stuck?
2. Did users have a clear starting point for how they should begin to complete a task?
3. Were users able to successfully complete the tasks they were asked to do?
4. What were your testers' expectations for what the assistant should be able to do? How did testers approach the assistant based on these expectations?
5. How do users structure their requests to the assistant—in short commands or longer paragraphs?
6. Under what conditions will your assistant be used? Will users be on a mobile phone or at a desktop? Focused or multitasking? Does this influence the way users interact with your assistant?
7. How do testers respond to the assistant's tone and voice?
8. How does the team publish the summary of testing sessions, to make the research accessible to others in the organization?

Review and Annotate Conversations

Review and Annotate Conversations

UX expert Jakob Nielsen is often quoted as saying that the first rule of UX is to base your product on what users do, not what they say. That is, you should consider what users self-report to be unreliable, and instead base your product decisions on user behaviors you observe.

With AI assistants, what users say is actually what they do. Software teams often have to infer and guess at user interactions by looking at heat maps or telemetry, but an assistant keeps a detailed record of every user interaction automatically, in its conversation data. Conversation data leaves nothing to the imagination—you can see exactly how every user interaction went.

Unlocking the power of this data is one of the most important things you can do to build better assistants, and it's one of the central ideas behind conversation-driven development.

Rasa X collects the conversations users have with your assistant and makes them accessible through a user interface. You can filter your conversations to surface interactions where a fallback action occurred, the channel the conversation came through, the length of the conversation, and more. You can also tag conversations to keep track of important trends.

But, conversation-driven development goes deeper than just uncovering insights about users. Conversations can be converted to training data for your NLU model, creating a virtuous cycle: users talk to your assistant and your assistant becomes

better and better at understanding what they say. With Rasa X, you can see which user messages don't yet exist in your training data. Then, you can easily label the intent and entities for the message and add it to your training data file. You can also use Rasa X to convert whole or partial conversations into training stories.

In this play, we'll walk through a few workflows you should know in order to analyze your conversation data and create training examples. Run this play with the members of your team who are responsible for adding new training data to the assistant and steering the direction of your assistant's design.

Play 3: Filter Conversations and Annotate Messages

Step 1: Filter conversations

Open Rasa X and navigate to the Conversations screen. First, we'll apply a filter to omit very short conversations. Open the filter menu, select Other as the filter type, and set the Message length filter to 5. This helps us weed out conversations where the user opened the chat session and immediately closed it. (Note that filtering for very short conversations can also be useful, to diagnose low engagement).

Materials:

- Rasa X instance, running locally or on a server
- A few conversations, collected in Rasa X. You can collect this data by running Play 2, or, if you're already in production, by connecting your live assistant to Rasa X

Time:

- 1-1.5 hours

People:

- 1-3 team members

Read through 3-5 conversations. Ask yourself:

- Did the assistant make any mistakes? What type of mistakes?
- Did the user's sentiment seem positive or negative?
- Was there anything surprising about the way the user phrased their requests?
- Was the user able to complete their goal?

Step 2: Use tags to apply labels

In the right hand panel, click the Tags gear and select the option to create a new tag. For each conversation you read, consider which labels might help you categorize and evaluate the conversation.

Here are a few ideas:

- Positive or negative sentiment
- NLU error
- Out-of-scope request (or feature request)
- Goal completed

Apply one or more tags to the conversation. If you find individual messages that require further discussion, use the message flag feature to mark and share them.

Step 3: Search for significant conversations

Return to the filters on the Conversation screen and note that you can now filter by the tags you've created. Use the filters to search your conversations and consider which produce an interesting sample set. For example, you might want to filter only by the Tester channel, to exclude conversations created by the development team, or you could filter for conversations where a fallback action was invoked or the NLU confidence was low.

As you read conversations, use the Mark as reviewed button to indicate you've read them (you can use this label to exclude conversations that have already been

reviewed from your filtered view). You can also use the Save for later button to set conversations aside if they seem notable but need further review.

Step 4: Annotate user messages

Navigate to the NLU Inbox. The NLU Inbox displays user messages that don't match any training examples you already have in your NLU data.

Use the sort dropdown to filter messages with low confidence to the top of the inbox. Working your way from lowest to highest confidence, correct the intent and entity labels if needed and mark the message Correct to add it to your training data. If you find messages you don't want to add to your training data, e.g. spam, delete them.

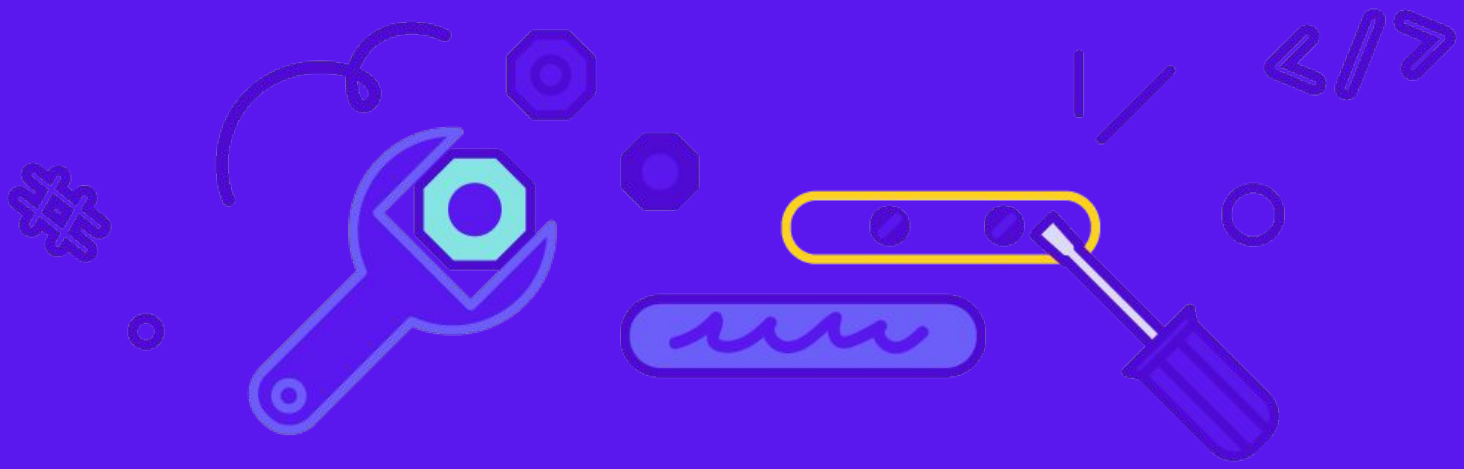
After annotating data, retrain and test your new model.

Step 5: Document what needs to be fixed

Create tickets for issues that need to be fixed, using your organization's issue-tracking or project management tool. If you're running Rasa X in server mode, you can include a deep link to the message or conversation where the issue was observed, to provide additional context.

Discussion Questions

1. Did anything surprise you about the way users formed their requests or interacted with your assistant?
2. Did users ask your assistant to do anything it hadn't yet been programmed to do? How did your assistant respond, and how could it respond better?
3. Which filters yielded the most interesting results?
4. Were there synonyms in user messages that your assistant hadn't seen before?
5. How do team members keep track of which messages have been reviewed?
6. Which roles on your team are responsible for annotating data? Which tools have they used for this task?
7. Did you find any user messages that weren't easily categorized into an existing intent label? How do you handle these messages?
8. Are there any patterns you see in the conversations that weren't successful?
9. How frequently are conversations reviewed by the team - daily, weekly, or monthly?



Fix and Test

Fix and Test

Reviewing conversations can uncover important problems: conversation turns that could have gone more smoothly, buggy custom actions, or phrases the assistant didn't recognize. The first step of reviewing conversations is to identify interactions where something went wrong; the next step is to identify how to fix the problem. That brings us to the next two stages of CDD: fix and test.

The approach you take to fix a problem will depend on the issue. If the NLU or dialogue model isn't making correct predictions, the issue likely lies in the training data. A good next step would be to check the training data for mislabeled examples, or add more training examples if the model is having trouble recognizing certain phrases. Or, you might observe that the assistant has trouble with requests that are out-of-scope or out-of-domain. If it makes sense to bring the request in-scope, you could build a new feature. If the request is something that you don't intend to address, providing a fallback or out-of-scope response can help get the user back on the happy path.

Once you've identified a possible solution, you can put a fix in place. That might mean annotating or restructuring training data, adjusting custom action code, or changing response templates. No matter which type of changes you make, it's important to follow engineering best practices like version control, automated testing, and code review to ensure updates don't introduce new problems.

Conversation-driven development hinges on small, iterative updates based on user input. Engineering best practices ensure that the constant flow of updates results in

real improvement (and it reduces the risk of regressions).

What does CI/CD and DevOps look like with a machine learning-based application? Full test coverage means evaluating the performance of the machine learning model in addition to the application logic. The Rasa CLI includes commands for running two important types of tests: conversation tests, which measure how well the model performs against a test set of whole conversations, and NLU tests, which measure the model's ability to classify intents and extract entities. When you run on a CI/CD server, these tests can tell you whether the changes you've made have resulted in better performance or introduced a new problem—before you deploy the new model to production.

In this play, you'll map out the pieces of your development and testing workflow and ensure that the process for moving changes from development to production is a smooth one.

Play 4: Establish a Development and Testing Workflow

Materials:

- Rasa X instance, running on a server
- Git repository for your assistant (Bitbucket, GitLab, GitHub, etc.)
- CI/CD tool (Circle CI, Jenkins, Bitbucket, GitHub Actions, etc.)

Time:

- 2-3 hours

People:

- 2-4 team members

Step 1: Connect version control

The starting point for your workflow is version control. Whether you're annotating data in Rasa X or pushing changes from your local IDE, your remote Git repository is the source of truth for your assistant's code. Establish the connection between your deployment and version control by using Integrated Version Control to connect Rasa X to your Git repository.

Step 2: Create a conversation test set

Create an conversation test set. Navigate to the Conversations screen in your Rasa X dashboard. Locate a conversation your assistant handled well—one where your assistant gave the correct responses to the user's requests. You want your conversation test set to be representative; that is, it should reflect the kinds of conversations users typically have with your assistant or a conversation you want to make sure your assistant continues to handle well.

Once you've located a successful conversation, click the Save test conversation button to automatically add the conversation to your test set. When you run the rasa test CLI command, the conversation test function checks the model's prediction at each step, so you can be sure a conversation that worked in the past hasn't broken.

Step 3: Set up a CI/CD build pipeline

Using the CI/CD tool of your choice, configure your build pipeline. Here's the basic recipe for deploying a Rasa assistant:

- Start with a base image that supports Python 3.6 or 3.7
- Install Rasa Open Source and any dependencies needed by your custom actions
- Run data validation, to check for mistakes in training data
- Train the new model
- Run conversation tests
- Test the NLU model
- Deploy the new model

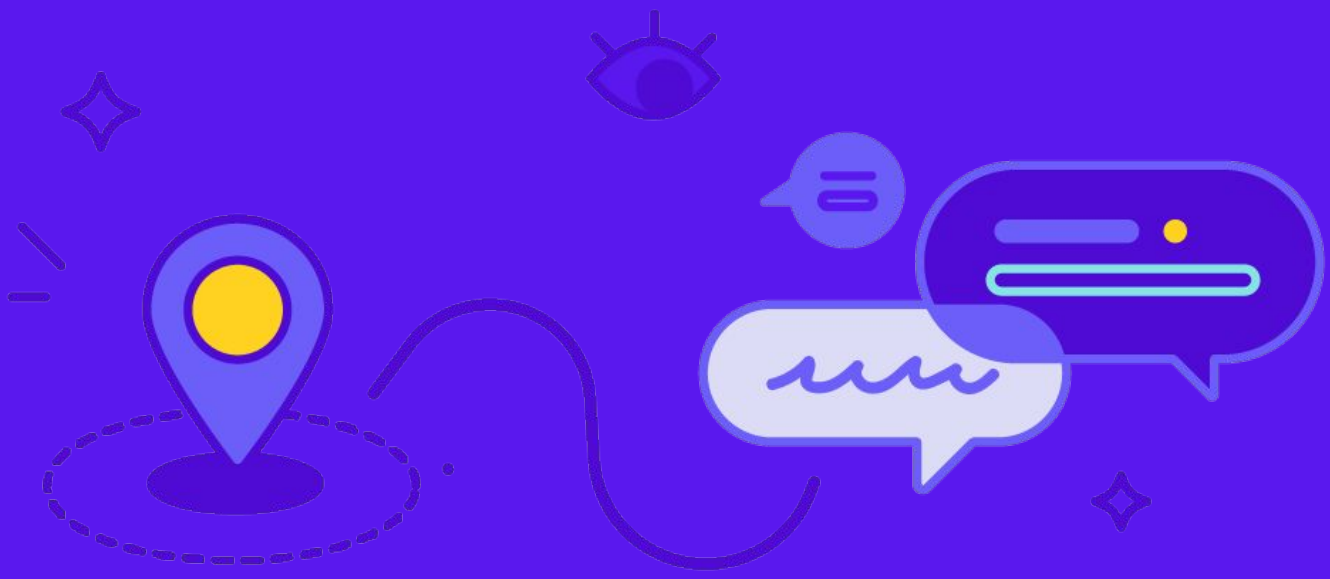
Step 4: Test the deployment

Trigger the deployment pipeline by making an update, for example, by annotating a few user messages in the NLU inbox. After annotating a message, you'll see the Integrated Version Control indicator turn orange; click it and push the changes to Git.

Trigger your CI/CD pipeline (you can configure which actions start the pipeline, but a common trigger is opening a pull request). Review the pull request as a team. Look through the test results produced by the CI run, and merge the pull request if the results check out.

Discussion Questions

1. What is the team's current process for making sure changes work in production?
2. Who is responsible for reviewing and approving updates?
3. What is the threshold for passing tests before a change can be merged?
4. How does the team approach writing unit tests for action code?
5. When might manual testing be required?



Track

Track

It could be the most important question conversational AI teams ask: is the assistant really helping users?

Teams typically look at two types of metrics to answer that question: direct measures and proxy measures. Direct measures are often what we think of as high level stats about the assistant and its performance. For example, teams can track the number of users and conversations the assistant handled, or the session length, or the average number of messages sent in a conversation. Digging one level deeper, you could also look at the frequency of intents to understand which requests or actions are the most common.

Proxy measures of whether an assistant helped a user are just as important, if not more. User behavior that indicates the interaction went well are often good proxy measures. For example, you might track whether or not a customer contacts support a second time within 24 hours, or clicks a link or CTA, or rates a positive NPS score from an exit survey. These actions don't necessarily happen within a session with an assistant, but they can tell you a lot about whether the assistant solved the user's problem.

Conversation-driven development emphasizes that effectively tracking success rates means looking beyond direct measures and top-level statistics. Proxy measures and insights learned from reviewing conversations provide a more complete picture of how well the assistant is meeting user needs. Too often, teams focus exclusively on top-level metrics and miss the understanding that can be gained from indirect metrics.

Use this play to align your team on the metrics that you should be paying attention to. Repeat this play in 3 months, 6 months, 12 months to reassess the metrics you track.

Play 5: Identify the Metrics that Matter

Step 1: Brainstorm questions

Ask participants to write down questions about the assistant's success rate on individual Post-it notes. Examples could include whether a user would choose getting help from the assistant over other methods, the number of contacts deflected, or how many users complete their goal within a session. Set a timer for 5 minutes while team members brainstorm questions individually.

Materials:

- Post-it notes and markers for each member of the group
- Whiteboard or large sheet of paper
- Timer

Time:

- 1-2 hours

People:

- 5-8 team members

Step 2: Rank the important questions

Collect all of the Post-it notes. The facilitator reads each note and places it on the whiteboard. Similar questions should be grouped together to track duplicates. Sort the questions according to which have the most duplicates—these can be considered "votes." Rank the top 5 questions according to the number of duplicate "votes" and group discussion. A question with few duplicates may be promoted to a higher rank if the group agrees it has merit.

Step 3: Establish measurable answers

Move all question Post-its except the top 5 to an out-of-the-way area of the board, designated the “parking lot.” Ask participants to brainstorm measurable ways to answer each of the top 5 questions and write down each idea on its own Post-it. Set a timer for 5 minutes while the team brainstorms.

Step 4: Balance direct metrics with proxies

On your whiteboard or sheet of paper, divide the space into two halves: Direct Metrics (measured inside the assistant) and Proxy Metrics (measured outside the assistant). The facilitator collects all of the Post-its from Step 4 and reads the ideas out loud. They sort each idea into the appropriate section of the board, depending on whether it’s a direct metric or a proxy.

When all ideas have been sorted, consider the balance of proxy to direct metrics. Are ideas skewed toward one area of the board or another? Discuss the outcome as a group and make sure the session has generated ideas for both direct and proxy metrics.

Step 5: Summarize your findings

Document the session by taking pictures of the board and summarizing the brainstorming session in a place where it can be accessed across the organization, like a company wiki. Identify next steps for instrumenting the tracking you identified in your assistant.

Discussion Questions

1. Is there agreement or disagreement about which metrics are most important across different roles?
2. For the metrics you've identified, do you currently have a good baseline for comparison?
3. What is the process for reviewing reports and translating insights into actionable tasks?
4. Which external systems and integrations might contain clues about how well the assistant is performing?
5. How do user goals and journeys contribute to selecting success metrics?
6. How do business goals contribute to selecting success metrics?

Conclusion

Users are the driving force behind every type of software development, and AI assistants are no exception. Conversational AI teams have a unique window into how users interact with an assistant, and successful teams channel conversation data into development decisions and model training.

But as we've seen, conversations are only part of the equation—you can't have CDD without development! And to that end, engineering best practices make up the other half of a comprehensive approach to building AI assistants. Teams practicing CDD work in short, iterative development cycles, using automated testing and CI/CD to ensure that updates are reliable and predictable.

Incorporating these practices into your team's workflow is a journey, and most teams today are already using many of these practices. The key is to view developing AI assistants as a partnership between your users and your development team, one that starts early and continues throughout your development process. With that mindset, your team can make the culture shift toward conversation-driven development and build a framework for creating AI assistants that truly help users.

CDD Checklist

Share

- ❏ Conduct a user test with internal volunteers
- ❏ Conduct a user test with a focus group of real users

Review

- ❏ Decide how often you'll review conversations
- ❏ In Rasa X, use filters to surface important conversations
- ❏ Identify issues that need to be addressed
- ❏ Identify successful conversation that can be turned into training stories and tests

Annotate

- ❏ Collect and annotate a minimum of 50 user messages
- ❏ Convert successful conversations into training stories

Fix

- ❏ Connect your assistant to version control
- ❏ Choose a ticketing tool to organize issues
- ❏ Make updates to address issues uncovered during review

Test

- ❏ Establish a CI/CD pipeline
- ❏ Ensure all intents, entities, and conversation paths, are represented in your test conversations
- ❏ Make automated tests part of your CI/CD process
- ❏ Institute a code review process

Track

- ❏ Identify proxy metrics as well as top-level statistics to measure success
- ❏ Use tags to label conversations

Additional Resources

- Webinar: 6 Steps to Conversation Driven Development (YouTube)
- Conversation-Driven Development Group (LinkedIn)
- Model Testing and CI for Conversational Software (Rasa Blog)
- Conversation-Driven Development with Rasa X (Rasa docs)
- Conversation-Driven Development (Rasa Blog)
- Write Tests! How to Make Automated Testing Part of your Development Workflow (Rasa blog)

What's Next?

Continue the discussion in the Conversation-Driven Development group on LinkedIn