EPISODE 1055

[INTRODUCTION]

[00:00:00] JM: Chatbots became widely popular around 2016 with the growth of chat platforms like Slack and voice interfaces such as Amazon Alexa. As chatbots came into use, so did the infrastructure that enabled chatbots. Natural language processing APIs and complete chatbots frameworks came out to make it easier for people to build chatbots.

The first suite of chatbots frameworks were largely built around rule-based state machine systems. These systems worked well for a narrow set of use cases, but they fell over when it came to chatbot models that were more complex.

Rasa also was started in 2015 amidst the first chatbot fever. Since then, Rasa has developed a system that allows a chatbot developer to train their bot through a system called interactive learning. With interactive learning, I can deploy my bot, spend some time talking to it and give that bot labeled feedback on its interactions with me. Rasa has open source tools for natural language understanding, dialogue management and other components needed by a chatbot developer. Overall, it's a chatbot platform that looks mature. It looks like a platform that has been worked on for five years and has reached a point where it knows how to give developers the tools that they need to actually build useful chatbots.

Tom Bocklisch works at Rasa and he joins the show to give some background on the field of chatbots and how Rasa has evolved over time. The hackathon platform I've been building called FindCollabs now has free hackathons for nonprofits and universities. If you are a company that's looking to sponsor a hackathon or if you are looking to run an internal hackathon, FindCollabs can also be used for those use cases. You can find all that at findcollabs.com.

[SPONSOR MESSAGE]

[00:02:01] JM: You probably do not enjoy searching for a job. Engineers don’t like sacrificing their time to do phone screens, and we don’t like doing whiteboard problems and working on

tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[INTERVIEW CONTINUED]

[00:04:18] JM: Tom, welcome to the show.

[00:04:19] TB: Hey, Jeff. Thanks so much for the invitation.

[00:04:22] JM: Chatbots, they reached peak hype three or four years ago. Everybody thought we were going to have chat bots everywhere, and then they fell off in terms of popularity. What happened during that cycle of chatbot popularity?

[00:04:40] TB: That's a really interesting thought. We noticed that as well, right? I mean, we started kind of during that hype. We created our first chatbots because of that hype. What we realized as well as lots of other people is that it is not easy to build a chatbot. It is not easy to build something that reacts well to users' inputs that can go beyond a single kind of a simple script following certain rules such as pattern matching. It's a lot harder to build something that is really meaningful, provides value and kind of add something on top of just a conventional UI or webpage or mobile application. That's actually a great resume and kind of the reason of why we restarted with the Rasa project. We wanted to make chatbots clever, and that's kind of the simple reason that we came up with the initial solution and open sourced that, got lots of feedback from the community. Now we're kind of seeing that people understand that there's a lot more behind building a chatbot than just pattern matching on simple phrases or kind of creating a rule-based tree to follow certain kinds of dialogues.

With the research that's now kind of on the way with all the research universities; Google, Facebook, all the new assistance coming out with Siri back in the day or Amazon Alexa, there is now a lot more focus again on building these assistance that really can make a difference when applied properly and developed kind of in a good way.

[00:06:10] JM: Right, because I think most people, when they think chatbots, the first thing they think about is you go to some webpage and there's little chat interface there. But if we scroll forward in the future a little bit, you can imagine the same sort of interface and the same sort of chatbot that you would make being applicable to Alexa or to another voice assistant platform today. I think even today, it is pretty portable, right? When we're talking about chatbots, are we talking about just a text-based interface or are we also talking about voice interfaces?

[00:06:48] TB: I guess you're kind of talking about both, right? There're lots more challenge you need to solve when you go for a voice-based assistance that you don't face when working just with text. For example, you're not running into the problems of needing to pick up voice, needing to pick up vocals. Having keywords and kind of finding good ways to – Knowing when to kind of listen to the user's input.

When you're just working with chat, that's a lot easier. As you said, it's a lot about texts, but building type chatbots can also include images, kind of generating graphs, using buttons. You have a lots of the user interface that you use normally to build webpages and you can implement them into an assistant as well providing kind of a user-centric experience.

Kind of regarding the portability, I'd say we're not there. Usually what happens nowadays is that you pick a platform and then you develop a bot for that. It's pretty easy to kind of port bots between, for example, the different messaging platform so you could create a bot for Facebook Messenger and then port that rather easily to WhatsApp. But it's a lot harder to kind of pull something on these platforms and import that to Alexa. We aren't quite there yet.

The usefulness for, for example, Alexa skills, is not quite at the level that we see for assistance that run on webpages, for example, providing help for user's in terms of IT support, for example, or helping them out with certain issues that they face when looking at the webpage.

[00:08:19] JM: The domains where chatbots actually have been working, what are those domains? What are the places where chatbots can actually solve a meaningful business problem?

[00:08:34] TB: Great question. We've been looking into that a lot, right? In the beginning, we've been focusing on a couple of verticals like healthcare, insurance. Basically, anything where you do customer support is a good use case, because there will always be cases that are hard to handle by just a machine or a clever assistant, but they will be suitable to handle the basic cases. Then from there on, you can expand them further, kind of reducing the load on people and your customer support staff. Letting them focus on the more kind of critical and complex cases and allowing kind of an automated assistant to handle the more easy ones.

More recently, I mean, given kind of the current time of SARS and COVID-19. That kind of in itself provides a really good example where chatbots can help a lot to inform people. There're lots of new regulations. There're lots more restrictions, and there's emergency programs, for example, to relieve companies and reduce unemployment and just getting to know all that information and finding your way through that information. Even if it's just kind of keeping up-to-

date on the latest news, kind of having a trusted source that you can ask questions, it's a great use case for an assistant.

We've actually been working together with the WHO health alert to kind of bring in assistance to WhatsApp that's used quite a lot right now to kind of stay up-to-date. Have a trusted source of information and kind of help people just navigate information and different urgently needed sources that – Yeah.

[00:10:11] JM: Let's talk a bit through the ways that the chatbot frameworks have been built in the early days during that 3 to 4-year ago period of time where chatbots were becoming really popular, and we'll gradually get to what Rasa does and what's different about that approach. If we think about that time when the major infrastructure companies; AWS, and Microsoft, and so on, we're building these chatbot frameworks, how were these frameworks implemented and what were the shortcomings of them?

[00:10:47] TB: They had built around the assumption that when you build a chatbot, you know all the different cases a user is going talk to the chatbot and you're going to know all the different kind of ways a user is going to follow. You know all kind of the dialogue paths the use is going to choose and you kind of are able to map that out as a large tree or kind of story. Let's call it a story graph. You're kind of really able to put every user on one of the kind of leaves of this large tree following one of the predefined paths. That is kind of the core idea behind all of these online services that allow you to kind of define some of the messages that you should be handling and then directly define the action they are going to execute.

Then nowadays, they improved that model a bit. So they're also taking some context into account so they can also react more actively to what's going on. But the traditional model has really been, "Here's the message," and then the bot puts that message into one of, let's say, five packets on it can either be a greeting, can be a goodbye message, or it can be, for example, restaurant booking message, and maybe the fourth, the fifth one are yes and no, affirmation and denying a request. You have five different messages the user can send, right?

Obviously, when you deploy that chatbot, users are kind of writing messages that you didn't think of. They're going to go conversational paths that you did not plan for. That's a huge

shortcoming people noticed in the early days of chatbot, which got them frustrated, right? I mean, you're talking to a chatbot. Maybe you made it half way through what you want to achieve, and then you write a message that the chatbot doesn't understand or didn't kind of plan for.

In that case, there is no way to resolve that with the existing platforms that were around a couple years ago. Yeah, you just need to kind of take more of these paths into accountants. What we kind of tried to go for is kind of learning from that data that you gather from users instead of taking that approach of having this predefined story tree, if you want to call it that way.

[00:13:03] JM: Yeah. The story tree, another way of defining this is state machine. Chatbots are often built with a state machine. The idea that the user starts and they're in one part of the state machine, and then based on some brittle rule set, the user is going to move through this state machine and gradually navigate to the place where they get their answer or the service accomplishes the task that they're looking for. Explain what the shortcomings of this state machine model are.

[00:13:43] TB: Right. The state machinery, I mean, itself has kind of a limited number of states, right? You can have only so many states. Transitioning between the states always means that you need to define what makes that transition happen. How do you go from state A to state B? How do you go from B to C? Then also modeling how do you go from A to C, for example. Coming up with these rules that you usually use, it's pretty tricky and it's darn hard to kind of maintain that over time.

Let's say you've created your initial state machine. Users are talking to that. It works fine for some cases, right? The bot runs through these states. A user writes a message. You transition to another state. Maybe calling and API, running some code, getting back to the user, transitioning in the new state, and now that fails. So the question now is what happens next?

Debugging these systems is really tricky, because when you're in a state, you need to figure out how did I get there? Then modifying the state machine to adapt to your new case where the state machine failed is pretty tricky, because you might be touching a lot of other states that you didn't think of or another path through your state machine that are triggered through other

dialogues and you might break them, and it's pretty tricky to kind of modify and maintain that over time adapting to kind of users' needs.

[00:15:11] JM: These state machines that are based on rules, you have a rule-based system. You can have edge cases that make that rule-based system exposed as being brittle. If you just think about the number of paths that a customer service agent would take, there are so many edge cases. There are so many different things that a customer could be looking for. One way to overcome the problems of this rule-based brittle approach is with reinforcement learning.

With every new user, you could try a slightly new approach for satisfying their needs. You could gradually train a model over time by giving it some kind of reward function based on how a task is successfully accomplished. How viable is reinforcement learning as a system for building conversational AI?

[00:16:09] TB: Very good idea, and we've actually tried in the past as well. I mean, we're kind of constantly looking into the ongoing research. We're trying out new methods, and reinforcement learning is something that we've been looking on especially in the early days. The issues that we saw when kind of adapting it and trying it out is that you need quite a bit of data to get started and also to retrain it.

The issue there is that the benefit the state machine has is it gives you certain guarantees. If you are in a certain state, you make certain input, you're going to end up in a certain other state. That's kind of the benefit of the state machine. Once you're kind of starting to use machine learning models to kind of predict what a bot or assistant should be doing, you're kind of leaving that certain certainty behind and you're moving into an area where it's a lot trickier to say what's going to happen next for certain.

That is one of the reasons why it's tricky to integrate that reinforcement learning, because it is hard to combine with these properties of being able to predict what happens next or defining in that sense of a state machine what certain properties should always be true, for example. What kind of things should not change? That's something we've been working a lot with. So combining these two systems in a way that you have certain properties that always hold and still

being able to use machine learning for the cases where you don't know exactly what should happen to kind of predict for good certainty what is a good reasonable next action.

In general, I'd say reinforcement learning is a good approach, but the issues we've found is that it just needs a lot more data than you usually have when starting out with a bot project and takes quite a bit of time to make that happen.

Another downside usually come on with reinforcement algorithms is that they can forget certain states and certain conversations over time. Once you've trained them and kind of retrain them over and over with kind of new insights, new conversations, as you said, some of that failed. You might run into issues where the algorithms forgets old conversations that have been working before, but it hasn't seen training data for them in a while, which is sometimes useful but sometimes it's a property that you'd like to avoid.

[00:18:35] JM: As we get to the way that Rasa works, could you define the term interactive learning?

[00:18:42] TB: It is kind of picking up on what you just said. How can we make sure that when a conversation goes wrong, we include that into our training data improving the assistant? Interactive learning is kind our solution or kind of a solution in general. When you have a machine learning assistant, you're talking to that. You're kind of observing what's happening, you now when things go wrong. For example, you know that when a certain property or state is hit, you know that the assistant did something wrong.

For example, the user just leaves in the middle of the conversation or a certain transaction failed. It's a good indication that something went wrong. In an interactive learning mode, what you're doing is you're talking to your assistant, and while talking to it, you correct it. How this works is you're really talking to it. The bot predicts what the input or the things to input means and also what it likes to do. Then you right there correct that bot to kind of do the right action, and then kind of the model gets retrained either directly or kind of after the session.

During that interaction of kind of this back-and-forth of correcting the bot, talking to it further, you're then able to kind of move the bot or shape it in a way that you want it to be in the way it's supposed to behave.

[SPONSOR MESSAGE]

[00:20:08] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its benefits, and we're happy to have DataStax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

DataStax provides DataStax Enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. DataStax Enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run DataStax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce. To learn more about Apache Cassandra and DataStax Enterprise, go to datastax.com/sedaily. That's DataStax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to DataStax for being a sponsor of Software Engineering Daily. It's a great honor to have DataStax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[INTERVIEW CONTINUED]

[00:21:47] JM: Rasa is based around actions. There's an action. An action could be to greet a new user or to give the user suggestions on good restaurants to eat at or call an API. The way that this bot is trained is through a process of interactive learning. Just to give people an example, let's say I want to make a conversational AI that I can talk to and get

recommendations about for which episode of the Software Engineering Daily podcast they should listen to. Walk me through that process of building my conversational AI. What would that look like?

[00:22:35] TB: Right. You would start with a simple initial bot. We provide kind of initial templates that just contain a couple phrases around greetings, goodbye messages, or yes-no, things like that. That would kind of give you the baseline for your bot. From there on, you would then start interactive learning session. You would start, “Hey,” and then the conversation would start. The bot would not know what to do next, right? The defaulting and the bot would do next is just listen, right? Would listen for another message.

Obviously, that's not the right thing, right? We want the bot to greet. We would correct the bot in that interaction and would say, “Hey, please run that action greet. Sending a message back to greet me.” We would correct that, retrain the model in the background. Run the new model, get the new prediction, and then we can move from there.

What we’re doing here is we’re kind of training that model to follow our first ideal scenario of greeting, greeting us back. Then we might ask, “Hey, kind of give me the best show of this year, for example.” Then we would start telling the bot, “Hey, now run my custom action. Predict that custom action.” The bot would do that, right? We can train it to predict that action, but we still need to implement kind of the API calls, fetching the data, getting the actual information of, “Hey, I've got the date range right.” For example, this year in this case, “Then fetch me kind of the most liked podcast.” That API is a bit separate and should be kind of – I don’t know, implemented in a separate service. Then you can call that from your bot. Integrating that information into the conversation.

For example, you might get back – I don’t know, a link for example or the title of that podcast. In your custom action, you would return that message back to the user proceeding in that conversation. In our interactive learning session, we would mock that interaction first, right? So we would say, “Hey, run this custom action.” That gives me back that URL or title of that podcast, and then we would proceed as if we would have had that information.

Once our interactive learning session is over, we would then actually starts and go and, for example, implement our custom action in Python. We provide kind of an SDK around that. It allows you to write your custom code in Python, calling our podcast API, and then we could combine these two pieces, right? We would have our custom logic written in Python and our machine learning model train based on our interactive session. We'd probably try it out again, right? We would, kind of as you go [inaudible 00:25:16] software development, you'd change it a little more. You kind of spin it up again, try it out. We would probably use different messages now in our second walk-through in our interactive learning session.

The bot might pick up our different messages in the correct way classifying it as kind of the correct intense picks up, or it might do something stupid and kind of predict something wrong. That's kind of ideal, because we can then use our new message to kind of retrain our machine learning model feeding it with a bit more information and more training data. That's kind of the model that we use when you kind of start out, want to create that initial version. The benefit of that loop as you can get started pretty quickly, and later on you cannot only use kind of sessions from interactive learning, but actual user conversations to train your assistant.

Kind of once you have that initial version that you're satisfied with that works in – I don't know, most of the cases you've tried. You would put that up, spawn it up on a server, host it for maybe, first of all, your friends to try out. They would talk to the bot. Probably some of the conversations would fail, which is great for you, because you can then use that data again to modify the model and train it rather easily. Once you're kind of satisfied and have a good feeling of the performance of the machine learning model, you would then put it life and then kind of get out to users talking to your bot. Then over time, you would observe the performance. How is the bot doing? Where does it fail? Are there certain interactions it commonly misunderstands? Then you would dive deeper again into single conversations using them to add to your training data to train your new assistant.

Exactly, that workflow is different from earlier chat bot systems, where you'd need to explicitly understand, "Okay, what is the underlying state of that conversation and how does that match with my state machine? Which transactions do I need to change? In our system, you just take the conversation. You label it correctly as another machine learning systems where you label

the data, and then you feed that into the machine learning training algorithm, and then hopefully next time, the bot handles that message correctly.

[00:27:26] JM: If I understand correctly, the way that this would work is I would create my basic chat bot for the software Engineering Daily bot and maybe I would test it locally in this interactive learning session and I would ask my own chat bot, “Hey, give me episodes about Docker,” and it might respond, “Look at these episodes about JavaScript, and here's an episode about entrepreneurship.” You could then in the interactive learning session let it know, “Hey, you did not respond correctly. You actually took the wrong action,” and you can update your model to respond more intelligently. Let's take that isolated example. Let's just say I have written my naïve chatbot and it's giving me bad recommendations. What am Is doing to improve its recommendation process?

[00:28:27] TB: That's a really good example, because when you ask, for example, show me podcasts for Docker, for example. An essential information you need to pick up from that message, and that is the part. The actual search the user wants to carryout. When you now run this in the interactive learning session, the bot will tell you what it picked up, right? It might not pick up that Docker part, or it might pick something else up from your message and misunderstand that, for example, for JavaScript or for query for that. That is your first point of contact where you'd look, “Okay, is that prediction correct? Does the bot understand that it needs to pick up the Docker part of my message? That is what we call entities. These are parts of messages that contain construction information.

Simple examples are numbers, for example, or dates, like tomorrow or next week, or like Docker kind of certain objects that we want to pick up in our query. If the bot doesn't pick that up, then it’s a great example to add to your training data, marking that Docker part as an entity. Telling the bot, “Hey, you need to extract this part of the sentence that we want to use in our custom action to then query the API using this Docker part as our search query for our podcast recommendation engine.

[00:29:49] JM: I want to understand to what extent natural language understanding is involved in this process. If the user asks some kind of question, is there a pre-baked model for understanding, like parsing the query and understanding what the user's intent is? How does

user's question get mapped to what the user is actually looking for and eventually mapped to an action that the bot takes?

[00:30:23] TB: Yeah, there's very different approaches to that. I mean, the simplest approach you can think of this just keyword lookups, right? So you could say if the message contains the search keyword, then that means for us, okay, the user wrote a message search. We're picking up the entity Docker and then we're executing the search. But obviously, looking just for single keywords doesn't really work in a lot of the cases. It's very cumbersome to come up with these keywords and to kind of maintain them when you have lots of these what we call intense, so kind of different types of messages.

Different approaches, and we've kind of recently integrated a new one into Rasa that we call DIET, and that's kind of a dual-intended entity transformers. Transformer being a specific type of machine learning architecture that we're kind of using as part of this algorithm.

The idea behind that is you feed in the different words of the sentence from the user one after another into that machine learning algorithm into the network. During the training time, we train it to embed these different words into a sentence embedding and compare that with the training data to see, "Okay, which sentence is our sentence that we posted to the bot? Which sentence is that most similar to compared to the sentences in the training data?"

This is an approach we've just released as part of Rasa. The benefit there is, in previous approaches, as I said, use keyword matching or maybe kind of what's called a bag of words approach where you don't take kind of the word order into consideration. That obviously leads to a lot of mistakes you make while picking up a user's intent. Kind of taking the order into account helps a lot.

I mean, you kind of point it to what parts of the software use NLP, and that is one part of it, right? Finding the intents and finding the entities. There're lots more processing that you can do that you can customize. For example, picking up dates as I said earlier, picking up numbers. That is actually done using regular expressions really advance. Once that we didn't develop ourselves, which is kind of another open source library, and that works pretty well for this kind of use case picking up certain numbers. But it doesn't work so well for kind of free-form entities like

our search for Docker. For that part, it's better to use the DIET algorithm that I just explained using transformers.

What we've noticed though is there're lots of problems that we can't easily handle with these machine learning algorithms right now. There is, for example, negation or nested entities, right? When you have an entity that's part of another one. So maybe you looking for NGINX Docker containers, podcasts, for example. Then you might want to extract NGINX as kind of a subpart of that entity. But also the whole thing, like NGINX, Docker container. That's pretty tricky to do.

Another example is if you want to define roles in your entity. Let's say you want to book a flight and you want to start at a certain airport and then fly to another airport. When a user says a sentence like, "I want to go to Berlin," then you don't only want to pick up a city, right? You don't want to only pickup Berlin as a city, but also that it's kind of the destination and not kind of the airport that you want to fly from. For kind of these tasks, you kind of need to use different algorithms. Kind of the benefit of Rasa is that she can plug and play kind of different components into your processing pipeline.

We provide the user a predefined set of components that we think are kind of a good choice, a good first start, and then you can customize that to fit your use cases if you, for example, come across another open source library that does certain tasks very well, for example, negation, then you can add that to your Rasa processing pipeline. You can still use all of our kind of tools to kind of train models in service app, persisting models, putting bots into production, but you can customize that very easily adding new features.

[00:34:33] JM: I'd like to go back in time a little bit to understand how Rasa came to the approach that has been so successful today. The company was started in 2015. You joined in 2017. Can you tell me what had happened before you joined the company? Where was it at when you joined and what's happened since then? Give me a brief timeline of how the company arrived at its current approach.

[00:35:04] TB: Right when I joined, we were about to open source the first library, which we called Rasa NLU, and that was focused on just classifying messages. We had a single message. You would say, "Okay, this is greeting, or this is a goodbye," and that was kind of the

initial library, right? No context. No dialogues. Just single message and ten mapping that to one of your pre-existing intents. That was kind of the state when I joined. We kind of polished that, open sourced.

I mean, to be honest, it was a small wrapper around kind of existing libraries, sklearn back in days, and spaCy for language model. We kind of used the existing tools and then apply them to that specific problem of intent classification, of message specification. Kind of providing users with first set of tools of kind of a standardized interface to make the building of an assistant easier. That was kind of the state back then.

Before we got there, Alan and Alex were working on kind of building bots on their own, right? I think the first bot they built was an analytics bot where a customer wanted insights about I think [inaudible 00:36:11] data and how many customers were visiting a website, and they wanted that in form of kind of a chat bot they could talk to and get insights out. I think when they started out, they didn't even create chatbot but kind of mocked that experience pretending to be the chatbot, and that led to kind of automating that more over time and in getting to that initial version of, “Okay, this is a library that we feel comfortable open sourcing,” and kind of getting into people's hands, getting them to play around with it.

From there, that worked more and more towards conversations. As I said, when I joined, it was just messages, single message specification. But what we noticed is that – I mean, you can't take a message out of context. You need to take a look at the whole conversation. What happened in the past? What other information do I have about the user to decide what to do next in a chatbot and assistant?

That's why we created what we call Rasa Core. This is the pride that looks at the conversation history and predicts our actions. This part decides what the bot is going to execute next. It can be a user. It can be a response to the user. It can be calling and API, things like that. This was kind of the second part in our journey. Then at some point, we decided to combine these two into a single Rasa open source framework for building contextual assistant.

This framework is on GitHub right now. I mean, we stay true to our path of kind of keeping all the things open source that you need to build a great and kind of mission-critical assistant. But

what we noticed is that these open source tools are great to kind of train models to handle the data and kind of persistence, spin up bots. But there's still an essential part missing, and that is that part of kind of diving into conversations when you have lots of them, right? Let's say your bot becomes more popular. How do you figure out which conversations went wrong? How do you know which conversations to look into to then convert them into training data, improving your assistant?

That is kind of the second product we built that we called RasaX. So that is a lot more focused on kind of diving into conversations and what we call closing the loop. So taking a look at conversations, it uses half with your bot, and then converting them into training data where it's necessary. Yeah, I mean, we've kind of made that freely available as well. We think users need that to build a good assistant. Yeah, from there, we've kind of expanded our products and we're working a lot on these features. Kind of making it really easy and understandable to build assistance right from the start.

[00:38:51] JM: Okay. If we revisit the podcast recommender chatbot, there're a few more concepts I'd like to explore. A developer can model this conversation with a story. Can you explain what the role of a story is in crafting a podcast recommender bot, for example?

[00:39:18] TB: Right. A story is what we call when a conversation makes it into the training data. When you have a conversation and let's say you've got a transcript of a conversation, then you know what the user said. You know what the bot answered. But what you also need to know is what happened in the background, right? Which actions did the bot execute? Which APIs did it call? What kind of response did it get? That is what we called a story.

A story is kind of a transcript in addition to all the actions that the bot executed in between writing messages and kind of all the external data it collected during API calls. Kind of representing that whole conversations and all the kind of what we call turns. So, kind of a single interaction with a user in that format of a story, and used that to train the machine learning model. Because when we want to train the model, we don't want to run the actions, right? When we process a user message, we don't actually want to call an API when we train your machine learning model. That is why we need the story, is to kind of represent the state that we actually faced when talking to the user.

In our example of the Docker search for our podcast, we would create a single story that would read something like, “Hello,” which is a greet intent. All that would be part of the story. Then we would list the actions, and the first section could be, “Hey, how are you?” Stuff like that, and what are you looking for. That is kind of the action that we would list in that story.” Then we would have another response from the user, for example, in our case, that could vary for the Docker podcast, which we would put in there as a message and also the annotated intent. Kind of the extra classification that we think that message should get. Then we would list the actions again that the bot takes.

It's always kind of this back and forth between user message and its intent, and then the actions of the bot. Kind of a story is kind of telling you what happened in that conversation including everything that happened behind-the-scenes, and that's super helpful for us to kind of then train a model based on that data.

[SPONSOR MESSAGE]

[00:41:39] JM: When I’m building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We’ve also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[00:43:28] JM: Just to drill further into that training process, let's say I do define the basic user story, like maybe the basic story is the user comes to my bot. The bot asks the user for some topics that they might be interested in, and then the bot uses those topics to generate a list of recommended episodes that the user might listen to. Then the user can continually say, “No, I'm not interested in these,” and give some more interests to whittle down their interests, and you can have a kind of loop there until the user is either satisfied or they leave. I can start off by – I take that story. I map out how my bot should work, in most cases, and then I have this loop that I’ve defined and I can just test it locally with this process of interactive learning. The interactive learning process I would be marking and letting the bot know that either their answers are good or their answers are low-quality and we need to retrain more. Over time, I’m going to train that model locally to handle a lot of the cases that it's getting wrong. Can you describe what is going on in the backend? What's the different software components that are involved in that story definition and feedback loop?

[00:45:04] TB: Great question. This is kind of what we've talked about earlier about the different parts of the software stack. When I explained that as Rasa NLU kind of analyzing single messages and Rasa Core looking at dialogues and kind of whole conversations, even though we merged them into kind of a single framework that uses just apply as is, these are still the parts that are kind of behind-the-scenes working to understand the user’s message.

When you create that training data, create these story files, adding more and more of them over time, what happens is that you feed the machine learning model more and more data. The idea behind that is the more edge cases you have as part of your stories, as part of your training data, the easier it is for the model to, in the real-world scenario, predict the right actions it should

take. We can't always be certain that it's correct, but at least we hope that due to the training data, it will do a better job at predicting what's next.

As soon as you kind of collect more and more data and kind of going through that loop more often, putting your bot into production handling lots of conversations, hopefully annotating some of them, the ones obviously that failed and would yield kind of the best results when getting annotated. Once you've collected more and more of these training data examples, you might actually invest more time into tuning your model. There are these two models to tune in kind of our default pipeline. The first one is looking at the single messages. So that's what we talked about earlier. The NLU part, the model that looks at what is the intent of that message and what are the entities in that message. That is the first model that you can tune.

You can either kind of if you're kind of more confident and working with machine learning models, you can come up with your own model. You can write your own Tensorflow model, reading in the data with our helpers and kind of pluck that into our pipeline, replacing our default model. We've also provided tools around being able to understand when a model performs well. So it's kind of evaluating which models are better and which are worse to help you in your search for a better model. That is kind of the first model. This is the NLU model, the single message model.

The second one is the one focused on the dialogue part. So what we call Rasa Core, and that focuses on predicting these actions that we want the bot to execute. That core model will take the user input in this structured formats that we got out of our first model. It will only see kind of the intent. It would only see the entities. It won't see the whole message. Based on that information from intent and entities and the history it sees from the conversation, it will then predict the next action. Pretty similar to the NLU model, you can exchange that model as well for other machine learning models either tweaking the models that we provide, modifying, for example, how many layers there are or how many folks you want to train for. But also using completely different models, for example. Plugging in bird as a model, or animal for that matter. These kind of features for customization are kind of a key thing why I think Rasa is successful and is so successful in attracting the kind of developers and kind of building a community, because we make it, and that's kind of one of our goals. We make it really easy to customize things as soon as you kind of need to and as you get more and more data as part of your bot.

[00:48:34] JM: Tell me about the Rasa business. Maybe you could give an example customer and explain how they're using Rasa and how that compares to the business, process of using the business compares to what they would get out of just the open source repository.

[00:48:54] TB: Right. As I said earlier, we have a strong stance on making things open source and freely available. We really want to make sure that everything you need as kind of a small team, single developer, anything you need to kind of build that bot, put that into production. We want to make sure that is open source and freely available.

What we provide to enterprises is tackling these more structural and organizational challenges. That means, for example, we've been working together with Helvetia, a large insurance company in Switzerland, and they have been working on a bot project. They're facing a lot more challenges within the larger organization. That means, for example, they need single sign-on to manage access to the data, and that's something that we provide as part of our enterprise product. Tackling these challenges that you face as a large enterprise. That's kind of the goal that we want to achieve of our enterprise product. And, we help them [inaudible 00:49:50], the debug that. Set things up in production. That's kind of part of our enterprise product.

[00:49:56] JM: Are there places where you notice in the customers that they simply can't get the chatbot to be as good as they want it to be? Maybe there's some recurrent theme in – Even the process of training a model, the interactive learning process and the experience that you've got to with Rasas today. Where are the places where it does not work?

[00:50:24] TB: Right. I mean, it's still early days in terms of natural language processing in general. There're lots of unsolved problems no matter if you're using Rasa or any other technical solution. There are still hard problems out there and there's lots of research going on to solve them. There might be cases and things you want to do in your chatbot that are just really hard to do right now with the current state-of-the-art. You just might be running into that, and we're really looking for that, and it kind of gives us good guidance on our internal research that we do and kind of what we need to look into next.

Also, as you mentioned, like using interactive learning, using stories. It is quite technical. Sometimes people just want to kind of spin something up, build something really quick, build a prototype. To be honest, right now, Rasa isn't the best tool for that. We're working very hard to make that a lot easier, but the learning curve right now is steeper than we'd like it to be.

This is something that, as I said, we're focusing on right now. But it's probably a lot easier to kind of create a markup and create kind of a simple thing to spin up with a different service Yeah. I mean, as soon as we talked about it earlier, as soon as you have more data and you want to improve that assistant, you really need that customizability of Rasa tweaking your assistant.

[00:51:39] JM: If you think about the percentage of queries that a chatbot could answer for an average company, like the insurance company the you mentioned perhaps, and you think about the percentage of coverage that the chatbot can answer versus the percentage that a human rep would be able to answer. If let's say in a situation where you've built a chatbot for some company, like an insurance company, and when the user gets frustrated, you delegate the work from the bot to a customer service representative, an actual human representative, what's the percentage of cases where the bot would actually be able to successfully serve the customer versus a human?

[00:52:30] TB: That's a great question and that's something that we're looking into very intensively with our customers, and it is hard to measure that in the first place, because you needs to find the conversations that went wrong to kind of measure that properly and kind of classify which conversations are kind of the good ones and which are the ones that we should have handled with a customer agent, for example. That depends a lot on the domain.

For that case, where we kind of working together with Helvetia, the insurance company, what they were doing as they were reaching out to customers to renew their insurance, kind of a state law, requires them to renew any insurance every two years. What they did before that before they employed a chatbot is they were just calling them by hand, right? That they would call every customer once their renewal date was up, and automating that was pretty easy in the first place, because it's a really limited domain. You have kind of a really defined goal. You can provide the additional knowledge the user might want to ask, like what is this service going to

cost or when is the next renewal or any other questions kind of around the policy is something that you could integrate into the assistant. Kind of making the experience for the user better. But still, it's a very limited domain, and therefore you're going to have lot higher success rates if you compare that to domain that's a way broader where you expect users to ask very different things and where problems get really complicated.

In these scenarios where you might be faced with very different user inputs, some of them might be easy to be handled by chatbots. Some companies estimate that to be around 20% and kind of the normal customer management. There's about kind of 20% that you'd say are right now easy to handle the with a chatbot, with automation. You're still need to make that transition from an assistant to a human. Make that really smooth for the user.

That can go in different ways, right? First of all, you need to have humans available in that case if you really want to do that. Then you want to make sure that you transfer as much information to the human chat provider or kind of your customer service agent as possible, right? Because you don't want them to ask the same questions your chat already did. Before handing of, it's also a good thing to kind of ask certain questions to kind of prepare the conversation for human taking over the conversation.

The great thing about our system is that you can then use these conversations where a human needed to jump in mark them and then kind of try to teach your assistant to handle these cases as well with the idea that over time you take over more of the parts that are maybe closer to the domain that you're already handling and taking off that load from your customer success agents and allowing them to kind of focus on the more critical parts of pulling them if things don't go the right way.

[00:55:27] JM: All right. Well, it's been great talking to you, Tom. I just want to close off with one last question. What's the hardest engineering problem that you've had to solve in your work at Rasa thus far?

[00:55:38] TB: Right. I mean, we're tackling a lot of machine learning problems that we've talked about earlier, but in terms of kind of more from the engineering perspective, I think the hardest problem we are facing is kind of bridging that gap between our uses that aren't too fond

of machine learning. I wouldn't say fond, but don't have a lot of experience with machine learning. They might not know all the algorithms.

Then on the other hand, we have kind of state-of-the-art research. We have new algorithms. What we are trying to achieve is we want to find a good abstraction of allowing our users who have little to no machine learning knowledge to use our framework and then over time maybe getting more comfortable, playing around with the machine learning, tweaking it here and there and finding a good abstraction there is really tricky. Providing the right layers, but also pulling in the user from time to time to kind of tell them, “Hey, you should maybe look into tweaking your machine learning model, because the configuration you’re using right now isn't the best,” for example. Finding that abstraction, allowing kind of users that are not proficient with kind of different machine learning models to use our software. I think that's been kind of the biggest challenge that we’ve been tackling from the engineering side.

[00:56:50] JM: Basically, the interface that you’re exposing to the users who – I guess, basically, the developers or the developers who may not be super experienced in machine learning. How do you give them the right interface to working with that machine learning?

[00:57:07] TB: Yeah, exactly. Because you want to make it easy to get started, to kind of pull people in, to provide good defaults, to provide things that work on lots of datasets. But then also in the future, it should be expandable. You should be able to kind of replace certain parts, kind of in a plug-and-play kind of fashion. It's pretty tricky to find a good level of abstraction there that is kind of suitable for most of our users. Yeah.

[00:57:32] JM: Cool. Well, tom, thanks from coming on the show. It’s been a pleasure talking to you.

[00:57:35] TB: Thanks a lot for having me.

[END OF INTERVIEW]

[00:57:45] JM: Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we’re being honest, I don't know of many engineers who really enjoy

building internal tools. It can be hard to get engineering resources to build back-office applications and it’s definitely hard to get engineers excited about maintaining those back-office applications. Companies like a Doordash, and Brex, and Amazon use Retool to build custom internal tools faster.

The idea is that internal tools mostly look the same. They're made out of tables, and dropdowns, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers can build these internal UIs in hours, not days, and they can spend more time building features that customers will see. Retool connects to any database and API. For example, if you are pulling data from Postgres, you just write a SQL query. You drag a table on to the canvas.

If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure. I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[END]