10

WeSay: A Tool for Engaging Native Speakers in Dictionary Building

Eric Albright and John Hatton

Payap University
SIL International

This paper introduces WeSay, an open source software application designed to involve language community members in the description and documentation of their language. Intended for rugged, low-power hardware, WeSay's simplified user interface removes many barriers that typically prevent the direct involvement of community members. In this paper, we describe the dictionary-building features of WeSay that allow a linguist to tailor a sequence of language documentation tasks to engage community members. These tasks reduce a production step to its simplest form, enabling focused training and division of labor. Word gathering tasks use semantic domains, word lists, or patterns of likely words to build up the dictionary. Successive tasks add specific content, such as glosses and example sentences, to the entries. In addition, the program can prepare simple paper publications designed to promote community support for the effort and can transfer the raw data to the linguist for further processing with tools that are more powerful.

1. INTRODUCTION[1]. Like many of the tasks of language documentation and description, creating a dictionary is arduous work. Yet the greatest resource, the members of under-resourced language communities themselves, often remain untapped. Our experience has shown that, when given appropriate tools, gifted community members are often eager to be involved and to contribute.

One might ask why it is desirable to involve community members in language work. On top of the evident rightness of helping interested community members play whatever role they can, we are struck with the fact that in some language communities, members are already attempting it. They may be diving in with inappropriate tools leading to a frustrating inability to finish, but they are doing what they can.

Recognizing an opportunity to expand the quality and quantity of language documentation collected by native speakers, SIL International and the Linguistics Institute of Payap University in Chiang Mai, Thailand are developing a computer application called WeSay[2]. WeSay is a simple dictionary-building tool for use by native-speakers of under-resourced languages. We have divided this paper into four parts. First, we give our perception of the current state of dictionary building by native-speakers. Next, we describe the personas that WeSay aims to serve. A list of the design goals of WeSay follows, and we finish with a walkthrough of WeSay and its configuration tool.

[1] The authors would like to thank the anonymous reviewers for their comments that helped to clarify the paper.

[2] WeSay is open source under MIT license. Downloads, videos, and source code are available from http://www.wesay.org

2. WHY ANOTHER DICTIONARY PROGRAM? As occasional consultants to those building dictionaries in Southeast Asia and Papua New Guinea, we have seen projects use a number of dictionary building techniques. In this section, we will look briefly at the advantages and difficulties of five of these.

As an easy way to get started, paper is very attractive. Dictionary Development Process (Moe 2006) workshops often produce tall stacks of paper. These may have a semantic domain written at the top, with column for vernacular and national language glosses. Difficulties arise when people want to actually do anything with all that gathered data. If they want to add more fields, correct the spelling, or make a publication, at some point, someone has to type it all in.

Other projects we have seen have made the very expensive mistake of trying to build a large dictionary by typing entries directly into a word processor in an approximation of published form. That is, they try to type in each entry using character formatting, as it looks in printed dictionaries. While this is easy to start, it is quite hard to finish. Different typists produce different formats, producing brittle documents that become unmanageable as soon as the font size or paper size need to be changed. When the programs they used (e.g. PageMaker) are discontinued and don't run on current computers, all of their hard work may be inaccessible. Recently, colleagues of ours have been asked to write a program to scrape the content out of a 20,000-word document featuring all these problems.

Other projects have used a humble spreadsheet. Each field is given a column. All the words that begin with a particular letter are grouped onto a single sheet within the document. This is a relatively reasonable approach for simple structures. This approach becomes unwieldy when the number of fields grows or more complex structures are introduced, such as multiple senses or example sentences. Complex structures are not part of the goal for many projects. The primary problem comes when the team desires to produce a publication. To the extent that the typists have been consistent in their use of columns, converting the system to one that can be published is not a big obstacle.

Systems based on Standard Format Markers (e.g. Shoebox) have long been a mainstay of dictionary development. While many consider this approach to be without significant fault, as consultants, we often tend to the weary and wounded practitioners of this approach, linguist and non-linguist alike. Some of the common problems we see include inconsistent use of cryptic backslash codes, multiple overlapping versions of dictionary files, lost settings files, a mix of Unicode and non-Unicode encodings within the same files, training requirements that can take weeks, and data stranded in one aging tool by a lack of consistently applied markup standards.

Finally, there are systems designed specifically for entering lexical data. These take responsibility for ensuring the data is well formed by presenting the user with forms to be filled out. Some of these are TshwaneLex (Joffe 2004) & SIL's FieldWorks Language Explorer (Butler and van Volkinburg 2007). These are quite powerful, but they are aimed at a very different sort of user than those we want to empower.

3. PRIMARY PERSONAS OF WESAY. In designing WeSay, we have found it useful to use what are called primary personas in the practice of User Interaction Design (Cooper 1999:137). A persona is an archetype who represents the actual users. Actual users will

have varying characteristics, but by meeting the needs of these imaginary persons, the software should satisfy a wide range of people. WeSay has identified two such primary personas.

3.1 ADVISOR. Our advisor persona is comfortable with everyday computer tasks, and can get occasional help from more experienced friends and consultants. She may be primarily working in literacy, translation, or lexicography and may be interested in producing language-learning dictionaries, or linguistic-oriented dictionaries. While she may not be a linguist, per se, she has enough training to guide the process of basic dictionary making.

3.2 USER. Our user persona is a native speaker of the language being documented. He lives in a village, but can travel to town occasionally and can meet with his advisor several times a year. While his educational opportunities have been limited, he is bright and the limited opportunities of rural life have not challenged his potential. He has no prior computer experience. What education he has had was in a second language, in an educational system that emphasizes rote learning rather than problem solving. While literate in the national language, he has rarely read or written anything in his own language, and so is unsure of spelling. He is motivated by a desire to help his people, and helped by regular encouragement, especially if his local community appreciates his efforts. The community can be a big help, but it can also act as an insurmountable obstacle to his work.

3.3 OTHER PERSONAS AND SITUATIONS. In taking this approach to software design and choosing these personas, we must consciously choose not to serve others. Though WeSay seeks to be of use to some groups, we make no claims about its usefulness to many or even most communities. For example, there are groups for which collaborative, web-based tools would be needed to overcome hurdles of distance and the desire for multiple people to contribute simultaneously. There are others for which there is a significant potential for dispute over the contents of the dictionary. This version of WeSay lacks explicit support for tracking who entered (or said) what, and what area they were from (though one can make custom fields to hold these kinds of data). If conflict resolution is needed, it will have to happen outside of the tool.

There are situations in which the would-be dictionary builder is not actually fluent in the language. WeSay could be of use to this person, but it is not designed with their needs in mind. Finally, there are important questions related to language revitalization efforts. How would a dictionary be used, if at all (Corris et al. 2002)? How might a tool be designed to encourage the widest possible community participation in the process of compiling a dictionary, under these circumstances? At this time, we do not have anything to contribute to these important questions.

4. GOALS. In designing WeSay, we had five major goals: (1) it should be simple enough to be used by people with little computer savvy; (2) it should support a task-based approach to dictionary creation; (3) it should keep the training load to a minimum by encouraging just-in-time training; (4) it should help promote community support for the process; and (5) it should run on inexpensive, rugged, low-power hardware. In the following sections, we consider each of these.

4.1 DESIGNED FOR NATIVE SPEAKERS OF UNDER-RESOURCED LANGUAGES. WeSay's primary distinction is that it has been specifically designed for relatively unskilled native speakers of under-resourced languages, empowering them to be active contributors and creators of dictionaries. Rather than taking already existing tools and stripping them down, we believe that tools that truly support these speakers must be designed from the ground up with their unique concerns and strengths in mind.

By design, WeSay supports only a subset of the task of dictionary creation. We have been able to keep it simple by omitting features, which, though they would be requirements for field linguists, are not requirements for native speakers. We view WeSay as a type of satellite application that can gather data. The data can then be processed by the more powerful tools if warranted by the goals of the project[3]. We would encourage linguists to continue to use tools such as Toolbox and Fieldworks Language Explorer for those aspects of lexicography that WeSay does not support.

Because many of these potential users of WeSay know little about computers, WeSay avoids the complexity associated with other software tools that have been designed primarily for expatriate linguists. For example, even computer savvy linguists often get confused and make significant errors when dealing with the computer's file system and file management tasks (such as finding, opening, saving and backing-up files). Someone may appear to do fine during training; but a month later, work has come to a halt. Users of Toolbox often report that they have lost all their work when, in fact, they just closed a single window and have no idea how to get it to open in their project again.

WeSay handles file management for the user automatically. When the user starts WeSay, he is back where he left off, editing the same file as before. Saving is constant and automatic. Backing up is both invisible and automatic. When the user desires to put all his work on a USB key, email it, or push it up on the web, a single click will do the job.

WeSay also has a very simple user interface. It avoids many common user interface conventions that add complexity, such as menu bars, dialog boxes, and many buttons. As for the data entry itself, the user is presented with forms to fill out, rather than being asked to enter cryptic backslash codes and maintain some abstract entry structure themselves.

To further simplify the user's experience, we locate all configuration controls in a completely separate program: the WeSay Configuration Tool. While still aiming to keep configuration easy, in this tool we put the full set of user interface widgets at the advisor's disposal. The advisor can rapidly switch back and forth between the Configuration Tool and the WeSay experience she is molding for the user.

Finally, we have kept the process and user interaction simple by focusing the user on a single task at a time. Each task has a user interface tailored to the job at hand.

---

[3] WeSay data is stored as a Lexical Interchange Format XML file (Hosken 2006) and can be exported to MDF (Multi-Dictionary Formatter) style Standard Format Markers (Coward and Grimes 1995).

4.2 SUPPORTS TASK-BASED DICTIONARY CREATION WORKFLOW. Another major distinction of WeSay is its task-based approach to dictionary creation. By task, we mean not only the user's undertaking, but also a software screen that maximally aids the user's productivity and enjoyment.

Traditionally, the task of lexicography has been tackled by working on single entries. Lexicographers fill out as much information as they can about a single entry, then move on to the next one. As new information is discovered, they may come back to an entry but the intent is to fill out an entry as completely as possible. In contrast, task-based dictionary creation focuses on adding only a single part of an entry at a time, but doing so across all existing entries in the dictionary.

Ron Moe (2001) describes a paradigm shift in which mass production techniques are leveraged by lexicographers. He suggested speeding up the process of dictionary creation with his Dictionary Development Process (Moe 2006) by using semantic domains to elicit words and then to expand the dictionary field by field. Those who have used this process have been excited about it (Shore and van den Berg 2006). WeSay's "Gather by Semantic Domains" task joins SIL's FieldWorks Language Explorer[4] in providing specific tool support for this process.

When we break down the big job of making a dictionary into the smallest set of steps, we are left with a series of tasks. Some of these would be:

thinking up words to be included in the dictionary,

differentiating the senses of the words,

glossing the words into another language,

adding examples to the senses, and

defining the words.

Many advantages come of this task-based approach. In the traditional model, people working on the dictionary had to have the training and entire set of skills necessary to complete any dictionary entry. With a task-based approach, the task can be performed by the person best suited for the job. Some tasks, such as glosses or translations, will require bilingual speakers, but many tasks, such as identifying words and providing example sentences can be performed by monolinguals. Different people may be involved at different times; one person may write examples and later another translates these. Because the bar is lowered, more people can participate.

---

[4] Like WeSay, Fieldworks Language Explorer supports Ron Moe's *Dictionary Development Process* by having a specialized input mode for gathering words by semantic domains. It also has a very powerful *bulk editing* tool (SIL International n.d.) which allows the user to automatically generate some fields based on the content of other fields (such as the tone pattern of the word, CV pattern of the word, or even the grammatical category for languages that have morphology that uniquely identifies the grammatical category).

4.3 KEEPS THE TRAINING LOAD TO A MINIMUM. By designing WeSay for native speakers and keeping a task focus, we aim to minimize the training required. For example, by hiding the computer's file system, we render unnecessary this major training area.

What really should go in each field? What is a definition, versus a gloss? The advisor can reduce training in terminology by translating WeSay's user interface into a language users can read, and can pick terms that will communicate well for them. This translation does not require interaction with the WeSay developers; the advisor does it using standard open source localization tools.

WeSay's task focus facilitates just-in-time training. A user can be trained only in the task at hand, rather than the entire dictionary creation process. The trainee is more likely to learn the material well and less likely to be overwhelmed by the concepts. He is less likely to forget what he was taught because he immediately puts it into practice.

4.4 ENCOURAGES COMMUNITY SUPPORT. We have designed WeSay with the understanding that the job of language description and documentation can engage not just gifted individuals, but the communities in which they live.

In many communities, someone with outside contacts and a computer will be knocked down, unless the benefits of the activity are seen to accrue to everyone (or at least the right people). With this in mind, WeSay supports making simple printouts of the dictionary and printouts of reports containing charts. If a printer is available, these tangible products can be shared in the community on a regular basis. By giving regular, smart-looking reports to community leaders, the dictionary builder can show respect for authorities, fostering both accountability to them and support from them.

One factor in community participation is simply whether or not the work is done within the community. When software is complex, users can quickly lose confidence, if not data. Thus there is a tendency to try to get the work done with computer help close at hand, in a large city or in a workshop environment. To the extent that a user of WeSay can be successful over sustained periods without such help, he has more freedom to do the work within the community, and over longer periods of time. Since the computer is taken to the community, more members of the community can get involved, whether by sitting with the WeSay user as he goes through tasks, reviewing printouts, or providing lists of words (on paper) grouped by semantic domains.

4.5 RUNS ON LOW-POWER COMPUTERS. Many people prepared to work on language development are located in rural communities. Many of these communities have dusty or salt-laden air, which shortens normal computer lifetimes. Many lack mains or even generator-produced power. Fortunately, it seems that these obstacles to community involvement are on the verge of being overcome. While it currently runs on Microsoft Windows machines, WeSay should eventually run on the coming generation of Linux-based laptops that are low-power, low-cost, and rugged. For example, MIT's One Laptop per Child project (OLPC) may well furnish a perfect platform for using WeSay in environments previously off-limits to computers. Costly hardware can still be a problem, even in situations where power and ruggedness are not an issue, so we are putting a lot of effort into ensuring that performance is still acceptable even using old hardware.

5. A USAGE SCENARIO. Let us take a look at how a community may develop a dictionary using WeSay. We have separate tools for each of the personas that we introduced earlier: the user, a member of the community who desires to help build a dictionary, and the advisor, the project manager who may be a member of the community or an outsider.

To get started, let us say that we are beginning with a blank slate, with no existing dictionary work to build on. The advisor begins by launching the WeSay Configuration Tool to create a new project.

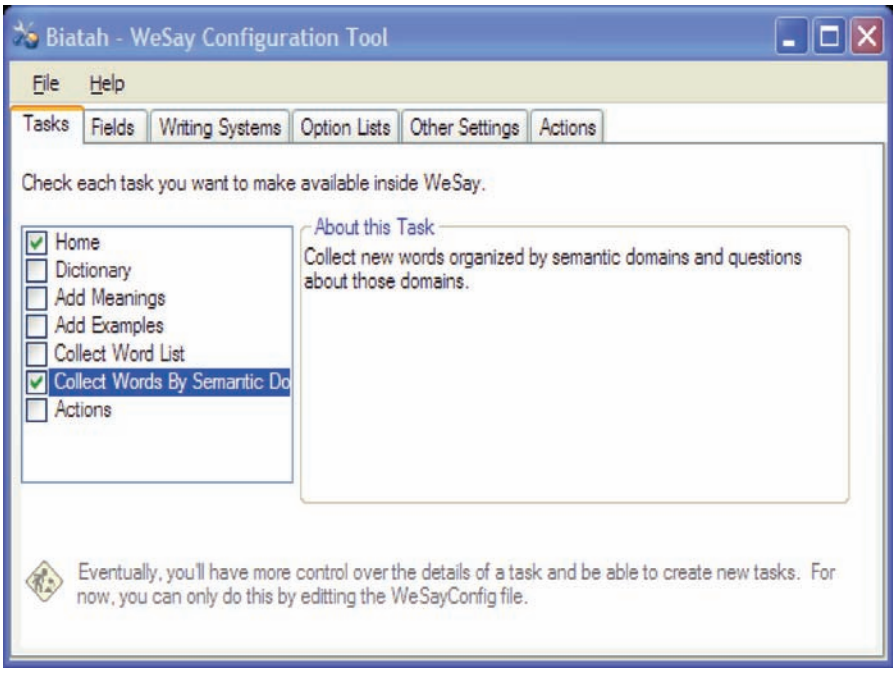

FIGURE 1

She sets up the writing system[5] information, and then decides that the first step will be to collect words using semantic domains. She turns on that task and disables all the others. At this point, she can launch WeSay to see what it will look like for the user (Figure 2).

5 WeSay has support for multiple scripts per language, including complex non-roman scripts. When we say "writing system", then, we mean a particular way of writing a particular language; such as Simplified Chinese, Traditional Chinese, and Pinyin.

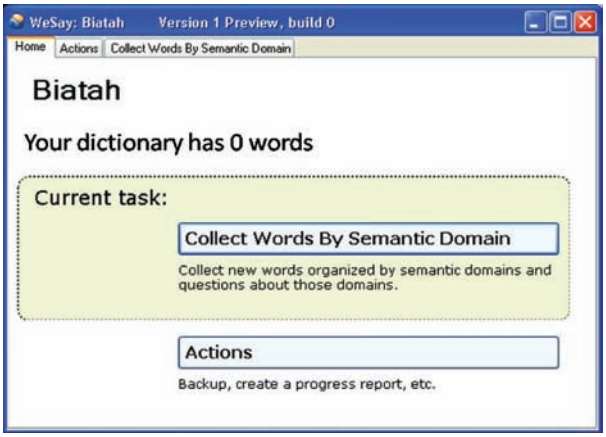

FIGURE 2

Satisfied, she trains the user on collecting words by semantic domains and they practice doing that using WeSay. She provides the user with a couple USB flash drives for backup, shows him how to plug them in, and which button to click on in WeSay to start the backup.

For the semantic domain word collection task, the user, perhaps in cooperation with a group from the community, thinks of words associated with a semantic domain (Figure 3).

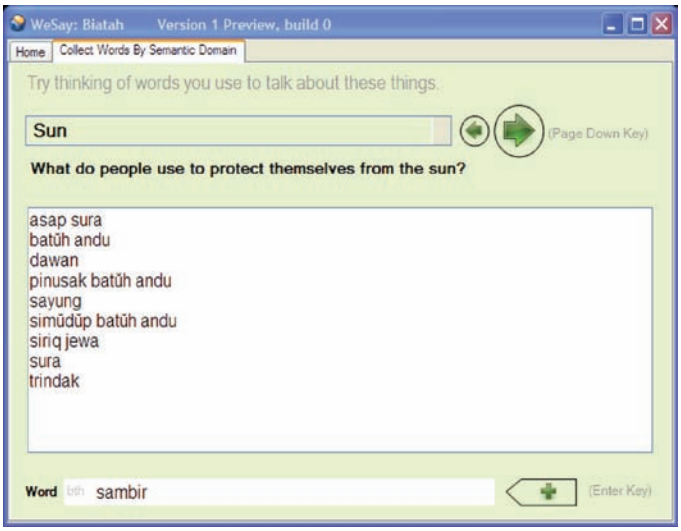

FIGURE 3

He does this by trying to answer questions that are designed to get people thinking about the domain. The user types as many words as he can think of for a question and then moves on to the next question. The words are associated with the domain itself, not the question; so as the user moves between questions, the words for that domain remain. When

the user goes to a new domain, the set of words is cleared and any new words become associated with this new domain. If the user needs to make changes to a word or remove it from the domain, he can select the word and it is brought back down into the editing area.

When the advisor is satisfied that the user understands the task and is confident with the tool, she can let him go until he has completed the task. Since version 4 of Ron Moe's Dictionary Development Process contains almost 1800 domains, this task could take a single person a few months.

When the user has gathered words from each domain, he returns to the advisor who can now set him up with a new task. Let us say the advisor decides the next step is to gloss into English all the words that have been collected. She brings up the configuration tool again and, this time, turns on the "Add Meaning" task. The user is finished with collecting words by semantic domain, so that task can be turned off if the advisor judges that the user is unlikely to go back to collecting more words that way.

The advisor may also choose to turn on the "Dictionary" task, which lets the user search for entries and edit them in place, add words, and delete them. Our advisor decides to have this task only display the meaning (gloss) field and the semantic domains of each sense (Figure 4), since any other fields would just be noise to the user at this point.

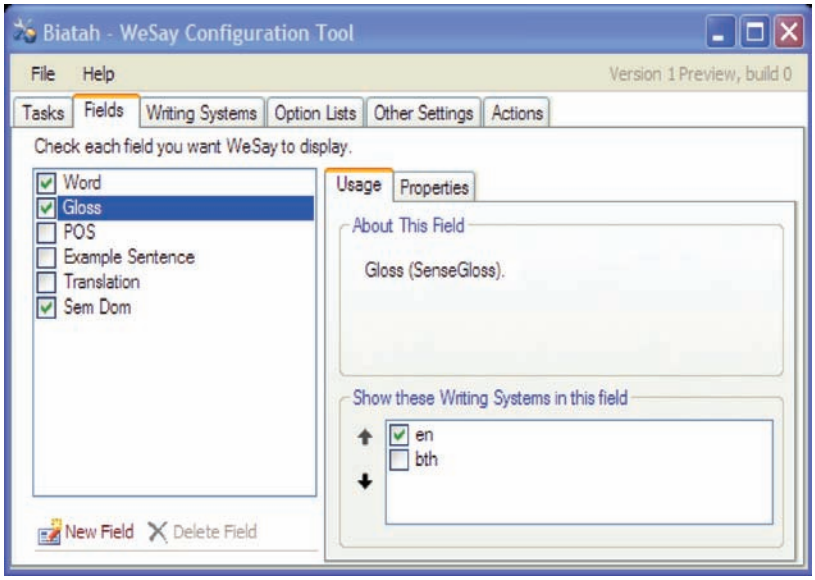

FIGURE 4

Now WeSay shows the "Dictionary" task, the "Actions" task and the "Add Meanings" task.

WeSay offers efficiency gains through task orientation; however, this does not limit the user to a single chance to do each task. There are two ways that a user can return to a task, after completing the bulk of the work in that area. Firstly, if the "Dictionary" task is

enabled, the user is free to add and edit information without using any of the more specific task-tools. Secondly, the dashboard that serves as the home screen for WeSay, enabled tasks indicate to the user when more work has piled up in a task. For example, if the user has added meanings for all the words in the dictionary, but later adds some new words, the "Add Meanings" task will display the number of new words that lack meanings (Figure 5). The number of items to be done in that task is shown in a graphic suggestive of an "in tray". This invites the user, over time, to return to tasks, keeping all of the entries complete.

FIGURE 5

Once again, the advisor trains the user for the work the user is about to do. Using the "Add Meanings" task, he can now work his way through the dictionary, word by word, adding a gloss to those entries that lack glosses.

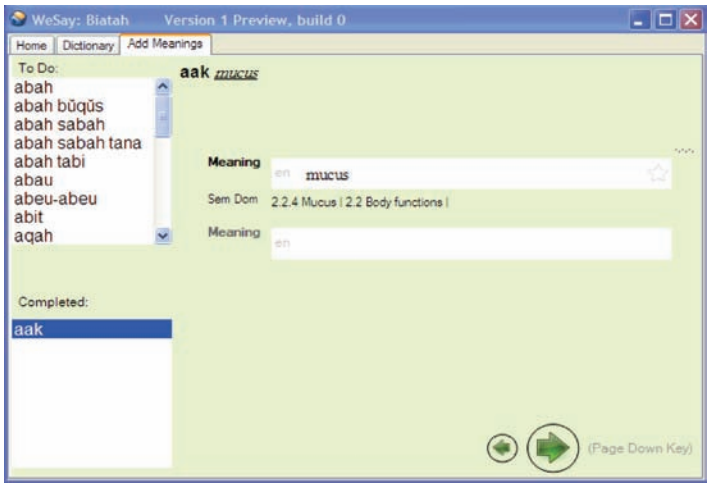

FIGURE 6

So what is the "Dictionary" task good for? Let us say that one evening, while sitting around the fire, the user hears someone use the word 'dog' in a way that he does not think is in the dictionary. Back home, he fires up the laptop and goes to the dictionary task. Before he can add a sense, he needs to find the existing entry for 'dog'. This may be harder than one would think.

Whereas many people take spelling for granted, in many of these situations, orthographies are still in flux and people may not have had a lot of experience writing their own language. This can make finding words difficult while providing ample opportunity for accidentally entering the same word into the system with multiple spellings. To help with this, WeSay suggests words that approximately match[6] what the user types.

Once the user finds the word, he adds a new sense by typing the new meaning in the blank space for the data. Just in case the user is not sure what will happen when he types in a new word, the preview at the top of the screen displays a blank to show where this new meaning will fit in the dictionary entry when it is printed (Figure 7).

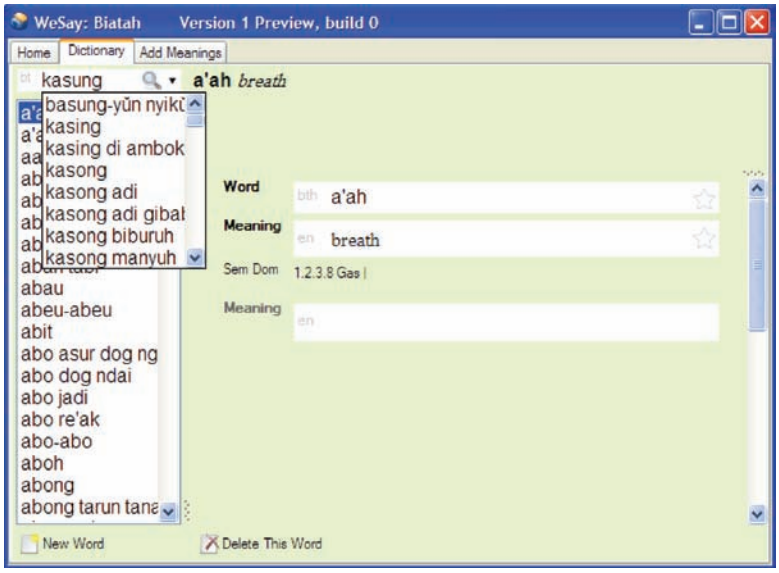

FIGURE 7

When the user has completed the "Add Meanings" task, he returns to the advisor who now enables the "Add Examples" task, and turns on the example field. Each time they come back ready to work on another task, the advisor progressively gives more training and unveils more features.

[6] WeSay uses a modified "edit distance" algorithm for this matching. While it works reasonably well in a wide variety of languages, we may provide the ability to tune it for particular languages in the future.

At any point in the process, the user can print a report that shows his progress or print a copy of the work he has done so far to give to community leaders and members (Figure 8).

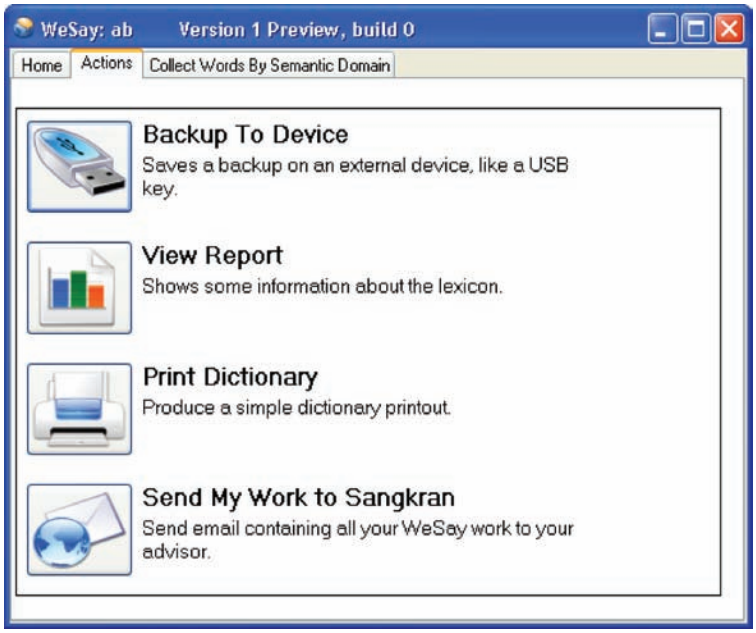

FIGURE 8

6. CONCLUSION. We believe that indigenous language communities need tools designed from the ground up, for their needs. We cannot just take software designed for expatriate linguists, strip down menus, and pretend that a product is now easy to use. By removing many of the obstacles that speakers of under-resourced languages face, WeSay enables people from under-resourced language communities to create dictionaries for themselves with minimal training on inexpensive, rugged, low-power hardware that can be used within their community. Thus WeSay can play an important role in enabling community members to be active contributors in the process of dictionary creation.

References

Butler, Lynnika and Heather van Volkinburg. 2007. Fieldworks Language Explorer (FLEx). Language documentation & conservation 1:1. http://nflrc.hawaii.edu/ldc/June2007/techreviews/butler.html

Cooper, Alan. 2004. The inmates are running the asylum. Indianapolis: Sams Publishing.

Corris, Miriam, Christopher Manning, Susan Poetsch, and Jane Simpson. 2002. Dictionaries and endangered languages. In David Bradley and Maya Bradley (eds.), Language endangerment and language maintenance. London: RoutledgeCurzon: 329-347. http://nlp.stanford.edu/pubs/eldic.ps

Coward, David E. and Charles E. Grimes. 1995. Making dictionaries: a guide to lexicography and the Multi-Dictionary Formatter (Version 1.0). Waxhaw: Summer Institute of Linguistics http://www.sil.org/computing/shoebox/MDF_2000.pdf

Field Linguist's Toolbox. http://www.sil.org/computing/toolbox

Hosken, Martin. 2006. Lexicon Interchange Format: A description. http://lift-standard.googlecode.com/files/lift_10.pdf

Joffe, David and Gilles-Maurice de Schryver. 2004. TshwaneLex – A state-of-the-art dictionary compilation program. In G. Williams & S. Vessier (eds.). 2004. Proceedings of the eleventh EURALEX international congress, EURALEX 2004, Lorient, France, July 6-10, 2004: 99–104. Lorient: Faculté des Lettres et des Sciences Humaines, Université de Bretagne Sud. http://tshwanedje.com/publications/euralex2004-TL.pdf

Moe, Ron. 2001. Lexicography and mass production. Notes on linguistics 4:3. ftp://ftp.sil.org/software/win/ddp/doc/ddp4_nlx_article.doc.

.Moe, Ron. 2006. Dictionary development process. SIL International. http://www.sil.org/computing/ddp.

One laptop per child. http://www.laptop.org.

Shore, Susan and René van den Berg. 2006. A new mass elicitation technique: The dictionary development program. Paper presented at tenth international conference on Austronesian linguistics, January 17-20, in Puerto Princesa City, Palawan, Philippines. http://www.sil.org/asia/philippines/ical/papers/vandenberg-mass_elicitation_dictionary.pdf.

SIL International. SIL FieldWorks Language Explorer features: bulk edit. http://www.sil.org/computing/fieldworks/flex/bulkedit.html

Eric Albright
eric_albright@sil.org

John Hatton
john_hatton@sil.org