We’d like to talk to you about how we’ve been building a visual editor for Wikipedia.
We are only 2/6ths of the VisualEditor team
Our team includes 2 engineers from Wikia – they also use MediaWiki
They also fight crime in their off time
There's also two remote people working on a new parser
This parser makes what we are doing with the VisualEditor possible
You might recognize this, it’s a Wikipedia article
You should edit it!
Seems simple enough, just hit the edit button and be on your way...
The Complexity Problem

Or not... What is all this nonsense you may ask? Well, it’s called Wikitext! Even really smart people who have a lot to contribute to Wikipedia find it confusing. The truth is, Wikitext is a lousy IQ test, and it’s holding Wikipedia back, severely.
The internet has normal people on it now, not just geeks and weirdoes
Normal people like simple things, and simple things are growing fast
We must make editing Wikipedia easier to use, not just to grow, but even just to stay alive
For the past couple years I’ve been absolutely obsessed with this problem. Obviously we need a way to make editing more like using a word processor. But after years and years of failed attempts, it was finally time to do it right.
First off, editing should be visually similar to viewing
Second, it should be clear what parts are text and what parts are objects
Finally, it should be easy to make things and hard to break things
Most important though, making an edit should be fun!
It should be fast!
It should be awesome!
The Complexity Problem

Well, maybe not that awesome.
I think this might be a problem.
You see, the reason Wikipedia is so accurate is because everything that’s changed gets reviewed.
The problem is it gets reviewed AFTER it’s already changed and made live.
Imagine a flood of edits begins to come in, and this is the user interface for reviewing them.
Balancing the ecosystem

It turns out that Wikis need balance
Balancing the ecosystem

If it’s easier to edit than to review than the wiki might die of corruption
Balancing the ecosystem

If it’s easier to review than to edit than the wiki might die of oppression
Balancing the ecosystem

Thankfully there are other teams at Wikimedia working on making reviewing much easier. The details of that however are a different talk.
Who here would consider themselves a Wikitext enthusiast
How would you react to someone taking Wikitext away from you?
Like taking guns away from Americans – have to pry it from their cold dead hands
And the truth is, it’s going to be a while before we have a full featured alternative
Theoretically when visual tools are equally capable they will be preferred.
To what extent?

Bringing the MAX to within 4 blocks of any point in town would be awesome, but impractical. We too will end up striking a balance, and some people will have to take the bus (click) Not every last feature of Wikitext will get the same level of attention, just the most popular ones. But as long as we can gracefully deal with foreign content, we can add new features over time.
To what extent?

Bringing the MAX to within 4 blocks of any point in town would be awesome, but impractical. We too will end up striking a balance, and some people will have to take the bus (click). Not every last feature of Wikitext will get the same level of attention, just the most popular ones. But as long as we can gracefully deal with foreign content, we can add new features over time.
So at this point, we don’t really know if, or when, Wikitext will go away completely. So we have to design around the reality that it’s here to stay for now.
Scale and speed

What happens when more people start editing faster than ever? More edit conflicts! Conflicts occur when the page is changed while you are editing. If our system can’t cleanly merge your changes, which is common, than you collide.
What happens when more people start editing faster than ever? More edit conflicts! Conflicts occur when the page is changed while you are editing. If our system can’t cleanly merge your changes, which is common, than you collide.
Currently, when there is an edit conflict, we try to merge the conflicting edits as single monolithic changes, and if there is any conflict anywhere, we bail out and let the poor user handle it.
What we need is a fully transactional system
Knowing not just where you ended up, but also how you got there, can make this better
We could even help solve the review problem by adding a playback feature
And also we can consider real-time collaboration, which merges changes as you type
Missing Pieces

Making editing easier is complex, lots of pieces have to come together. We are focusing on just one piece, and working closely with a team who’s focusing on another. A visual editor this is not a silver bullet, many things must come together to solve this problem properly.
Let’s talk about Wikitext
Like any markup, it uses special sequences of characters to describe
Structure (click), text content (click) and formatting (click)
People invented it because it’s relatively easy to read and write, at least compared to say...
HTML, everyone’s favorite markup language
While this is commonly written by hand, it’s not optimized for that
It’s not optimized for visual editing either as it turns out
But this is.
What you are looking at is a JSON serialization of our linear data model
It’s what our editor is thinking about while you are selecting and typing
It’s even more verbose, so we when we are using a whiteboard it looks like this (click)
But this is.
What you are looking at is a JSON serialization of our linear data model
It’s what our editor is thinking about while you are selecting and typing
It’s even more verbose, so we when we are using a whiteboard it looks like this (click)
The important part about this format is how easy it is to:
Select (click), delete (click) and insert (click) data
The important part about this format is how easy it is to:
Select (click), delete (click) and insert (click) data
It’s especially superior to HTML when selecting arbitrary ranges (click)
And then trying to delete (click)
This format also makes it possible to use linear transactions, let’s go back
It’s especially superior to HTML when selecting arbitrary ranges (click)
And then trying to delete (click)
This format also makes it possible to use linear transactions, let’s go back
It’s especially superior to HTML when selecting arbitrary ranges (click)
And then trying to delete (click)
This format also makes it possible to use linear transactions, let’s go back
It’s especially superior to HTML when selecting arbitrary ranges (click)
And then trying to delete (click)
This format also makes it possible to use linear transactions, let’s go back
What we actually did to the document can be described as 3 discrete operations (click)
We retained 13 items (click), replaced the selection with nothing (click), and retained to the end (click)
A transaction processor applies these operations to produce the new document (click)
To reverse this, we can simply flip the operations (click), and process again (click)
This is more than undo and redo, it opens the door to rebasing, playback and realtime collaboration
What we actually did to the document can be described as 3 discrete operations (click)
We retained 13 items (click), replaced the selection with nothing (click), and retained to the end (click)
A transaction processor applies these operations to produce the new document (click)
To reverse this, we can simply flip the operations (click), and process again (click)
This is more than undo and redo, it opens the door to rebasing, playback and realtime collaboration
What we actually did to the document can be described as 3 discrete operations (click)
We retained 13 items (click), replaced the selection with nothing (click), and retained to the end (click)
A transaction processor applies these operations to produce the new document (click)
To reverse this, we can simply flip the operations (click), and process again (click)
This is more than undo and redo, it opens the door to rebasing, playback and realtime collaboration
To keep a structured UI in sync with a linear model, we need a node tree
We build it from the linear data, and then build a user interface from there.
We also store lengths in the node tree so finding offsets is fast.
This structure is also very efficient when inserting or removing content
- Once the linear model is changed (click)
- A document synchronizer updates the node tree (click)
- Then the user interface responds to events emitted by the node tree (click)
This structure is also very efficient when inserting or removing content
- Once the linear model is changed (click)
- A document synchronizer updates the node tree (click)
- Then the user interface responds to events emitted by the node tree (click)
Early on we had a theory:
- Content editable might get you up and running fast, but it also limits what you can do
- Google Docs took this route as well, which gave us some confidence
- It appeared that doing everything ourselves was possible, we called it EditSurface
- This turned out to work pretty well, and we solved a lot of tough problems
A text-flow algorithm can be a tricky thing to write. Using a `div` for each line requires measuring the line each time a word is added and breaking the line when it no longer fits. It's also gotta be pretty darn fast.

The solution is to manually flow text into
- Flowing rich text into individual lines
A text-flow algorithm can be a tricky thing to write. Using a div for each line requires measuring the line each time a word is added and breaking the line when it no longer fits. It’s also gotta be pretty darn fast.

And since we are doing this on our own, we had to retain support for floating elements.
A text-flow algorithm can be a tricky thing to write. Using a div for each line requires measuring the line each time a word is added and breaking the line when it no longer fits. It’s also gotta be pretty darn fast.

And since native browser selection was a nightmare we had to render selection with divs. And to capture input properly we had to use an offscreen focused input box.
A text-flow algorithm can be a tricky thing to write. Using a div for each line requires measuring the line each time a word is added and breaking the line when it no longer fits. It's also gotta be pretty darn fast.

And since native browser selection was a nightmare we had to render selection with divs. And to capture input properly we had to use an offscreen focused input box.
A text-flow algorithm can be a tricky thing to write. Using a div for each line requires measuring the line each time a word is added and breaking the line when it no longer fits. It's also gotta be pretty darn fast.

And since native browser selection was a nightmare we had to render selection with divs. And to capture input properly we had to use an offscreen focused input box.
A text-flow algorithm can be a tricky thing to write. Hi. It's also gotta be pretty darn fast.
This was awesome, it felt native, and it made our laptops happy. But mobile devices were sad, they needed lots of native support we couldn’t get – Like selection, spell check, auto-complete, auto-correct, etc.
Content editable is necessary

2 members of our team revisited this theory and made some breakthroughs. We developed both versions in parallel, and after a month we changed course. We still fight content editable every day, but the awesome native features are worth it.
The trick is to make use of native goodness
– But revoke the browser’s decision making capability
The trouble with ContentEditable is that it’s essentially an unpredictable black box. You give it content as HTML, let the user modify it with a keyboard and mouse, execute some limited commands, and then hope the HTML that comes out is sane.

**Hint:** it won’t be – If the user so much as presses enter, your document is going to be trashed.
The trick: A custom model and a view and controller that abstract ContentEditable
The most difficult part of this approach is observation
– Some systems are eventless, like spell check, autocorrect, or drag and drop
– The events that are provided rarely contain enough information
When handling events, only some are useful – they will lead to model and view changes. To fill in the gaps, we must periodically check to see if something changed:

- When you notice a change, you can then update the model.
- It can still be tricky to know when it’s safe to re-render.
- Especially with input method editors, which have their own state.
http://www.mediawiki.org/wiki/VisualEditor:Demo
What’s next?

We have a long way to go, but we’ve architected the system for enhancement over time.
We are also now working on an easy to use API for adding functionality to the editor.
More Features

- Nested lists
- Definition lists
- Tables
- Images
- Videos
- Infobloxes
- References

- Image galleries
- Real-time collaboration
- Conflict resolution
- Edit playback
- Integration with discussion system

We have a long way to go, but we’ve architected the system for enhancement over time. We are also now working on an easy to use API for adding functionality to the editor.
We have also been working hard to reduce dependencies on external libraries and systems. This editor is at its core, an HTML editor, and we want people to use it everywhere.
Get Involved

Learn more about VisualEditor

http://www.mediawiki.org/wiki/VisualEditor

Clone our repository

git clone https://gerrit.wikimedia.org/r/p/mediawiki/extensions/VisualEditor.git

If you want to get involved, check out our wiki
You can also clone our repository
Work @ Wikimedia

http://jobs.wikimedia.org

Wikimedia is also hiring a variety of positions
For more information, checkout jobs.wikimedia.org
Trevor Parscal
trevor@wikimedia.org
@trevorparscal

Roan Kattouw
roan@wikimedia.org
@catrope

Learn more about VisualEditor
http://www.mediawiki.org/wiki/VisualEditor

Download these slides
http://wikitech.wikimedia.org/view/Presentations

Thank you!
Any questions?