웹 소프트웨어 신뢰성

✓ **Instructor: Gregg Rothermel**
✓ **Institution: 한국과학기술원**
✓ **Dictated: OES 봉사단**

Alright, so last time we looked at one instance of a piece of a research relative to a class of web applications, and this is another example.

Whereas last time we were looking at web macros, which are a form of web application, very specific and a rare task, and now we are looking at something called mashups.

And mashups are ways to assemble a bunch of, or to grab data from a bunch of web sources and integrate them and do something with them and create some results.

Now they are meant to be primarily something that non-professional programmers could do.

You could obviously write some sort of application that integrated multiple sources of data from different web applications using higher level languages, but people who don’t have that programming skill sometimes want to put things together as well.

And that’s what mashups are meant to allow, although professionals can use them too.

And so there are, as the paper says, there are several different interfaces available to create mashups.

Does say that there’s somewhere, yeah, now, I haven’t checked lately to see which of these are still available.

IBM mashups center, I believe, is, but that you pay a fee for.

Uh, the other ones, I am not sure.

I mean, Yahoo pipes is available, definitely still.

And there may be other new ones since then, but there are one way for putting data together.

Maybe they are Poor Man’s Service Compositions, I don’t know.

There are Poor Men, that is another English saying.

A poor men’s steak, a hamburger is a poor man’s steak.

Does that make sense? Mashups are poor men’s service composition creation device.

So the outline of this talk, which pretty much matches the outline of the paper.

Involves some motivation, some info on the related work.

The discussion of what this work is about, which is a versioning system, empirical study, and results and conclusion.

So as I said, web mashups, I have a cold, so I hope I can talk loud enough, I hope I won’t lose my voice, web mashups pull in data from multiple sources, and put them together through one interface in order to display them.

And then the environments that make them, such as Yahoo Pipes, provide repository to users.

The IBM provides repositories to users who pay for it, where you can store your mashups and run them.

So the mashups end up being run not on your machine, but on some central server.

And the results and you activate them to a client and the result gets displayed through a client.

Now, what this, this paper, being about versioning, you all know about versioning having connections to software engineering and building software that we use configuration management systems can create multiple versions of our software and save those versions, um, and that helps in a lot of things.

And has been looked at for a lot of things.

If you are new in a project, you can look back at the old versions.

You can see how systems evolved and why.

You can trace changes back to try and see when a change was made, particularly if it caused a bug, and that of course leads to debugging.

Um, in general, versionings beneficially to maintenance because it lets us keep track of the work we do on the system.

And the mashup programming environment doesn’t have any concept at all of versioning.

So far as any we have been able to see.

All they have is a repository mashups, and if you are user and go in and edit a mashup, it edits that mashup.

And the old version is lost. And now, our theory, or thesis was that by providing some form of versioning to the people who use mashups, we could provide them some benefit, which would help them create or understand or debug mashups better.

And here is some of those statements.

We think you might be able to create them better, they might be able to, it might make mashups more reusable, might help users understand evolution.

And we wanna do these without making them learn things like check-in, check-out, well, having to learn soft engineering concepts so much or having an interface where they have to type commands like we do.

So this ended up being the first attempt, we know of to create and study versioning in a mashup environment.

There’s been some attempts in other user programming environments, but not regards to mashups.

So there was some related work on this, um, some of the problems people have already been discussed in prior work, and they include problems that think could be helped by versioning.

Rosson and Zang looked at some other issues about types of information users run into.

They studied web users to see what they might be able to do with or find in mashups. So those are good papers to read….

And so those are good papers to read to get some back up, back ground but there’s not much else beyond that in mash up at least the time that work was done.

Now to study yahoo, to study mash ups, we needed an example of a mash up environment and again the paper lists several, we had to choose one and we chose Yahoo pipes A.

Because is commercially available and has been used quite a lot and that means there are lot of existing mash ups in there and it’s least somewhat representative of a mash up environment that people find usable.

Important for us also that it was for free, we were able to get a one week license to use the IBM one for free but that’s all we could get.

So we chose Yahoo pipes for that reason too and in looking at it, this is one of these things that was a risk affront that we had to make it sure, we had to make it sure that we've had way to get at the data because the Yahoo pipes sever belongs to Yahoo pipes we can't get in the code for that.

We had to be able to get into the data and do what we needed to do in some way without having access to their code and we found that we could by a matter that I show, a picture of in a minute and here was Yahoo pipes mash up and, how many, have anyone of you used Yahoo pipes?

At this point? Ah ok. When you read this or sometime before? You read this earlier, didn't you? Ok ok.

So it's not that hard to use, I don't think. Well, for us particularly it's not so hard to use. But there is what they call a canvas, like artists, There is an artist canvas is what they put paint on.

So is a canvas. And there are, and you select from a list of available modules, modules that perform like think of these ApIs, they provide certain sevices.

You select from list of those, you can drag them on the screen, and then you connect them up with what's called the wires, and then you parameterize them.

So here is a search module, or you can put in search terms, that's a search and this is filtering out, filtering out unique, non unique items, and this is sorting them.

So that's a pretty simple function grabbing a bunch of stuff that meets a search term, filtering it, and sorting it and giving you the output and the parameters here item dot title descending for order.

But you can bringing in data from lots of sites and scan it for information on various things stock prices, reports on companies, that kind of thing, Now we want it to put versioning in this, as I said, there is the Yahoo pipe server and there is the client.

And normal people are building their pipes on the client or essentially what they see is building the pipe on the client, but the work is going on the server, and when the pipe runs, it's on the server and we want it to wait to save information and so we implemented essentially a proxy that's sits in between the two that intercepts the messages between them, and that can take for instance, when a user submits or does something with the pipe, it can take it and to save that in the database and keep track of what version it is.

Now we could've used the version management system like Get hub or CVS or something but for this we just use a simple data base, it worked out fine. But this lets us keep versions and keep track of the versions and now we can change the interface in the client, as you will see in a minute and use this information in addition to whatever comes from yahoo pipes.

Now in our system, the versions are generated whenever users saves or clones a pipe, a clone you probably read, happens when the pipe, the repository over on Yahoo has a bunch of pipes in it, created by various people, if I want one of these, if I want to reuse one of these, I can do what is called cloning it, grab it, make my own copy, exact copy and now I can edit it and diverge it from that ok.

So that's how you make copy but or you create one from scratch or you save this actually cloned it, so whenever you save a clone a pipe we'll create a new version of it.

And then our extension which is on the client side, lets the user go back to earlier versions and what's happening really is you've got there are actually versions of them here, the earlier versions are actually yahoo pipes as well.

And we add some widgets to the client that aren't their normal client which are undo, redo, and histories of pipe pipes, and the history in of the pipe things will show these different versions of the pipe that exit and there's some special things here like the ability to indicate a base line which is something that's thought to have to reach a stable point.

And what are listed here are information on modules in the pipes we redo our versioning at the level of the modules.

So this is a little table from the paper contrasts that let's say professional configuration management systems like CVS SVN with our extension. So think of the professional systems as working typically your own JAVA programs, C programs really any file.

For programmers it's programs.

The versioning unit for low systems is usually the file.

Individual files. For us the versioning unit is the module.

And you can have a pipe with the various modules and we'll keep track of the different stages those modules can be in.

In the CM system, you specifically create a version by committing, by checking in. Here, we don't want them to have to do that, so that says I said any time they save or clone a pipe, we will create a version. And they can go back.

Browsing activities in CM systems you can undo or select a version.

Here they can undo, redo, or select something from the list view.

So basically they can go anywhere back in the tree, but undo and redo just move one step.

And those are common operations for users.

You can create a snapshots in the CM system, you can specifically make something a base line by conglomerating together different parts of the system.

And here by running one successfully you can create a base line it'll be marked as such.

CM system's allowed differencing, in professional ones they tend to difference in the files.

Here's version one of file one, here's version two of file one.

We'll do differencing on the level of the individual modules.

Deltas, you can add delete and modify in those systems, we just do add and delete or modify essentially a delete followed by an add here.

Merge exists in those and we haven't tried that yet.

And our population is different than theirs. So we implemented the versioning system.

And then set out to begin to study it.

And this first study is a relatively small a think aloud study where we're trying to get information on what people think and what effects they find that happen when they use the system.

And a think aloud where it's one on one, it would be like if I brought each of you in to work with the system for an hour, you verbalized what you were doing, that gives us a lot more information.

That noise is awfully annoying isn't it.

Is the window closed? Thank you.

Now an alternative would have been to start with a complete controlled study.

Get twenty people to use the versioning and twenty not.

But there we wouldn't have the opportunity to interact with them and get feedback which is useful at an early stage.

So, we're going to ask does it help people perform tasks related to mashup creation?

And does it help them understand complex mashups.

These were two things we were thinking in the beginning that versioning would help them do.

So we ended up getting nine students, now this is not a very diverse population because they're all male and they're all computer science or engineering.

So pretty much as in the first study as in the macro paper, we can't generalize much from this.

But we did feel we could get some useful initial from this information.

Four of them undergrads, five of them are graduates.

None had used yahoo pipes, but all had done some programming.

So this study here is what we call, a single factor within subjects design and we will look at that more in the next couple classes but the factor is the independent variable it's the thing that vary which you're looking at like a medicine and another medicine.

In this case it's using yahoo pipes with versioning and using yahoo pipes without versioning.

That's the one factor.

Within subjects means we have these nine people.

I talked about this a little yesterday.

With nine people I could say, okay four of you are going to use the control technique and five of you are going to use the experiment.

And I'm not going to let those who do this use this and I'm not going to let those who use this do this.

And so I'll get some data on how they did and some data on how they did. And that's called between subjects.

Within subject says well I'm going to let all nine do both of these.

Now there're pluses and minuses to that.

Letting nine do both, means we can ask them well what did you think of this versus this.

You did them both.

What did you think? Also with the small number, say let’s get more data on the two approaches.

Than if we split them up.

But it also has potential for what we call merging effects.

If I give them all this first, they’re going to learn a bit more about yahoo pipes at that time and when they get to this they may do better.

If I give them all this, it could be the same way.

So, that’s where a between-servers broke them up could do better because we got people who only did the one and who only did the other.

But then, we really want larger groups.

But, we chose between primarily, because we wanted to get their feedback on both of these and cause us small group.

And we have them all to control first, because we wanted them to do things without any versioning.

And then we had them do the experiment, and the one with versioning.

So, the first thing we had to do is that, since they had used pipes, well, actually, the first thing was the background-questionnaire so we know who we have there and we look on back on that later.

And then we gave them a tutorial.

We gave them a chance to create a pipe.

And we gave them their two research questions.

There were the two research questions as we recall.

We gave them a pair of tasks.

For the first one, which was, can you break a pipe?

And a pair of task for the second.

Can you understand one? And then we gave them a survey.

This was, again, one person at a time.

And it says here something, maybe.

I guess that’s much later. Well, okay, it says in the paper.

This was done in a usability lab in university Nebraska, where we can bring them in and actually record what they do, and keep transcripts that you can later go through, and that kind of things.

So this is just a picture of what’s going on with the tasks.

First task on the left, we use ability, second on the right.

And let me look at the right first.

The second tasks learn ability, that’s just controlled by experiment.

Where we try to learn or understand ability.

But the first task breaks down a little different.

And the first task they do control on experiment.

But there were two parts.

And each of these, the first of all understanding the pipe they’ve been given, and a second involved creating, essentially a modified version of that pipe.

How to do that twice.

So, there. Oh, sorry. On the side, it was the ten module pipe.

We gave them one, and they could use that and modify that to grade a new one.

On the right, it was a much larger pipe, and what we were doing was quizzing them on its functionality after the expense of some time understanding it.

Both of these- I don’t remember why.

I was trying to get rid of that. I don’t know why it does that.

I couldn’t get rid of that. So there it was.

We talked about threats to validity last time.

We always want to analyze those.

External validity means how the results generalize, well, we only have nine participants.

They were all male, they‘re all mostly CSE students.

So, that’s a pretty restricted group of people.

And we couldn't say much beyond that.

We only had two tasks. So, that's not a huge set of tasks to look at.

So it's hard to generalize very far from this.

But we are more interested in getting data on things to improve the approach.

So, that was okay. Internal validity? Again, that's thing that might cause us to have different results that aren't independent variable.

So, possibly, if they weren't fully concentrating or something, they might do different on the two tasks.

That might lead the results.

There is the learning effect thing.

The thing we didn't do was counterbalance the tasks.

There were, I'm pretty sure that we all have them do task.

One, first task and two second.

We could have split those. Some do task and two first do task to one second.

But we didn't do that.

Construct validity.

We have a control task, and experiment task, and we're trying to measure the differences between the approaches.

And if the tasks aren't of same, comparable difficulty, we may not get the measurement correct.

So, we break the research questions down into more precise hypothesis.

The first one, these are stated as 'alternative hypotheses', I think I said the last time that the experiment may start with the null-hypothesis, which is the opposite of the what you think you can prove, and that's what does statistical tasks start to done on.

And if you can reject final hypothesis, you can assert the alternative.

But you can state the alternative, just stats tasks done on.

So, the first concerns, remember first to look for creation, and less time is required to create a pipe when using versioning, than the versioning isn't there.

And pipes with versioning are more correct than those without.

And also, concerned the first research question.

And then, the second, we are interested in whether we can help them learn more, or understand those larger pipes they were looking at in that.

Better with versioning.

So, what’s with the first type hypothesis?

We start by looking the total time spent to create the pipe.

Remember. Time is the first issue, and correctness, is the second.

And there were two parts to creating it.

You might remember.

There’s some time that took them to understand, and then they started to making changes.

And we separate that out here in these bar charts, and what does chart showing is, for the participants, hang on, oh.

They actually tend to begin with the second one we had to drop them for some reason I no longer remember what it was.

So, 1-E is participant one under the experimental using the versioning, experimental treatment. 1-C is person one under the control treatment.

And so, 1-4 is going across.

And this is showing then understanding time and creating time.

And so the first thing we wanted to do was, did having this affect the time? You can look at this and say oh that’s less than that, in total as a total bar that’s less, that’s less about equal less, less, tiny bit less, less, less, ok?

But is it statistically significant?

That means are these results that could be had by chance or are the results really attributable to having the difference in tools so that’s where you use a statistical test and we use the Will Coxcon signed rank test is appropriate in this case and the p-value the value for which you can reject the no hypothesis, the p-value was point 0.6 and usually we say 0.5 is the threshold beneath which you can reject alternative hypothesis.

And so this says there is a statistically significant difference in the time required to create the pipe.

Looking at this set of nine persons, so there is a difference that just say statistically significance difference.

The next question was the source of these differences so look at this also highlighting going on that I could have been doing but that's ok.

We break that down into two sub tasks understanding and creating so here we did that and this is just looking at the participants in the control task in the experimental task.

The time they took to understand the pipe and here you see quite a few differences control took more there took less there.

More there less there, more less, less so kind of all over the place and this statistical test didn't give us any statistical significance.

So it doesn't look like the use of versioning was helping them understand the pipes better.

So why were they creating them faster?

Well it has to come down to the time spent implementing.

Once you understand it you need to change it in some way and that's where having versioning made those with the experimental task.

Visually you can see quite a bit better and that came out to be statistically significant too.

So all and all we can say it takes less time to create when they have versioning or at least in this study I need to qualify this in this study with these people.

With these tasks it took them less time to create things and that was due to implementation time not so much understanding time.

Correctness.

Well in this case how are we going to judge correctness?

First off this is going to show some correctness score on a scale of a 100% for the people's pipes.

There are the 9 participants with number 2 being left out.

Now these correctness scores how did we get those correctness scores?

We had two people two are the authors first and third authors look at the pipes independently of one another and judge them on a grading scale that had been created before the experiment proceeded and then they got together and came to a consensus on the grades and so for many cases everyone got perfect scores whether they were on control experiment in 4 cases.

Here you got equivalent but imperfect scores and there are one two three four cases, where the control task got lower sometimes bit substantially scores and so we did a Will Cox test on that and that actually we were really surprised we did get statistical significance even though there is only 4 cases where it's there.

So it is showing that the result isn't just due to the random chance, ok?

But in terms of the magnitude of difference just looking at the numbers maybe not quite so much there.

Then we will look at the second hypothesis this was on much larger pipes where all we were asking to do was understand what was going on with those pipes and then fill out and then answer some questions about them, and we scored their answers percentage of correct answers and here again experimental task in red we got the same percentage, same percentage control not quite as good, some percentage experimental not quite as good.

Same, same so it looks like almost as many cases where one's better than the other and there we did not get statistical significance at all.

Between the behaviors.

So again keeping in mind that this is one small think a loud study of nine people from a limited without much variance in their background and doing just a couple of tasks at least in this study we observed diversioning helped people create new pipes more efficiently and more correctly but it did not help in learning and the paper does talk about several things that are like qualitative analysis when you look at think aloud.

You can analyze the data and talk about what the people were seeing and what they experienced and just a couple of points from the paper people were finding that without versioning locating on the original and the apparent pipe was problematic and so as they started to change things they wouldn't be able to roll back so the simple thing there is you starting to make changes to a starting point and they would have a hard time retrieving the starting point when they realize when they have been going the wrong way.

And that's related to the second one we just maybe go too far or move too many modules and it would be hard to go back so there is more things talked about in the paper, so this is really a restatement since it's a conclusion.

Versioning improved time and correctness.

Reusability we don’t know about learning so obviously we now want to go farther than this try larger samples and hopefully true end user programmers.