

Crystal FLOW

for Better Results in Less Time

Do you fully understand the code you are working on ?

Answer: If you understood the code **fully**, then there would be no bugs in it.

By increasing your **understanding** of the code,
you can increase the **quality** and **correctness** of your code.

To have *a better understanding* of the code in less time - *Use the right tool for the right task*:

For example:

1. To review a function and the functions called by it, view its [CallFlow](#).
2. To find why a data object contains an incorrect value, examine its [Data-dependency tree](#).
3. To understand a function quickly, view its [Flowchart](#).
4. To examine how a data object is used in the project, view its [DataFlow](#).
5. To view the calling relationship of functions within a file, view the [File's function tree](#).

and so on...

Crystal FLOW

for Better Results in Less Time

By using the right tool for the right task,

you can have a better understanding of the code in less time.

It will increase the **quality** and **correctness** of your code.

Suppose you are making a **design change** that affects a particular structure member and you need to examine **how that structure member is used** in the project.

Use the right tool: View the DataFlow of the structure member.

- ◆ **The DataFlow** shows:
 - all the places in the project where that structure member is used
 - and the control flow surrounding those places
- ◆ Its graphical view gives you a **clear understanding** of how the data is used **in much less time**



Crystal FLOW

for Better Results in Less Time

Design changes have better quality and correctness when you use Crystal FLOW.

- Before the design change, you can **analyze** the existing code and affected data **in less time**.
- After making the design change, **check for correctness** with DataFlow, Flowcharts etc.

Team discussions are quick and more productive with Flowcharts, Rich Trees, DataFlow, etc.

When you **inherit** legacy code, Crystal FLOW will save you a lot of time.

Code Reviews are **quick and productive** with HTML documentation containing Flowcharts etc.

The graphical views of Flowcharts, DataFlow, CallFlow enable you to **think at a higher level**.

Everyday, you spend many hours in browsing, understanding and editing code.

With highly effective tools such as: **Flowcharts, Rich Trees, DataFlow, CallFlow**

Data-dependency trees, Premium browsing, etc.

You can save more than 1 hour every day!

Crystal FLOW

Making Better Design Changes in Less Time

◆ **Before** making changes,

Crystal FLOW will help you analyze the existing code and affected data objects in less time.

A few examples:

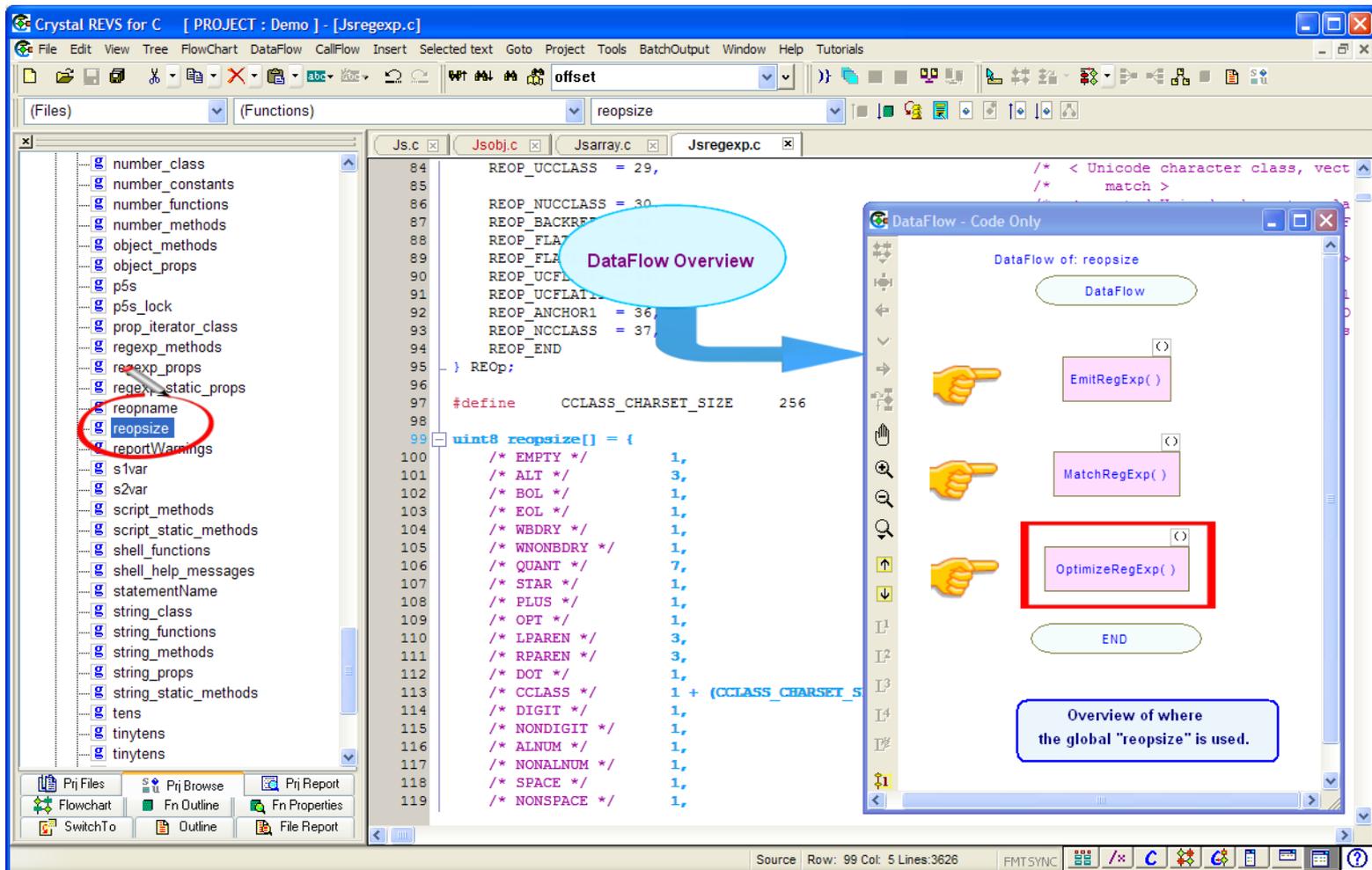
1. **How a global or any variable is used** in the project (Use DataFlow)
2. **What functions are callers of** a given function and **what functions are called by** a given function. (Premium Browsing)
3. **What parameters are being passed** in the call-sequence up to the current function (Use Rich Tree)

◆ **After** making code modifications:

- **Check the modified code for correctness** (Use Flowchart, DataFlow)

(every problem you detect will save much time later)

1. Use DataFlow to examine how a global variable is used in the project



Here the DataFlow Overview shows: the global variable reopsize is used in three functions - namely, EmitRegExp(), MatchRegExp(), and OptimizeRegExp()

Double-click to expand "OptimizeRegExp()":

1. The DataFlow expansion shows: **how the global variable** reopsize **is used in** OptimizeRegExp().

The screenshot displays the Crystal REVS for C IDE interface. The main window shows the source code for `Jsregexp.c`, with lines 1689-1690 highlighted in yellow:

```
1689 state->progLength -= reopsize[
1690 state->progLength += reopsize[
```

A callout bubble points to these lines with the text: "How 'reopsize' is used".

The `DataFlow - Code Only` window on the right shows a control flow graph for the `CALL OptimizeRegExp` function. A box labeled "DataFlow of reopsize" is connected to the graph. The graph includes a "Tracking: reopsize" box, a "case REOP_CCLASS:" block, and a "NO" branch leading to a yellow box with the following code:

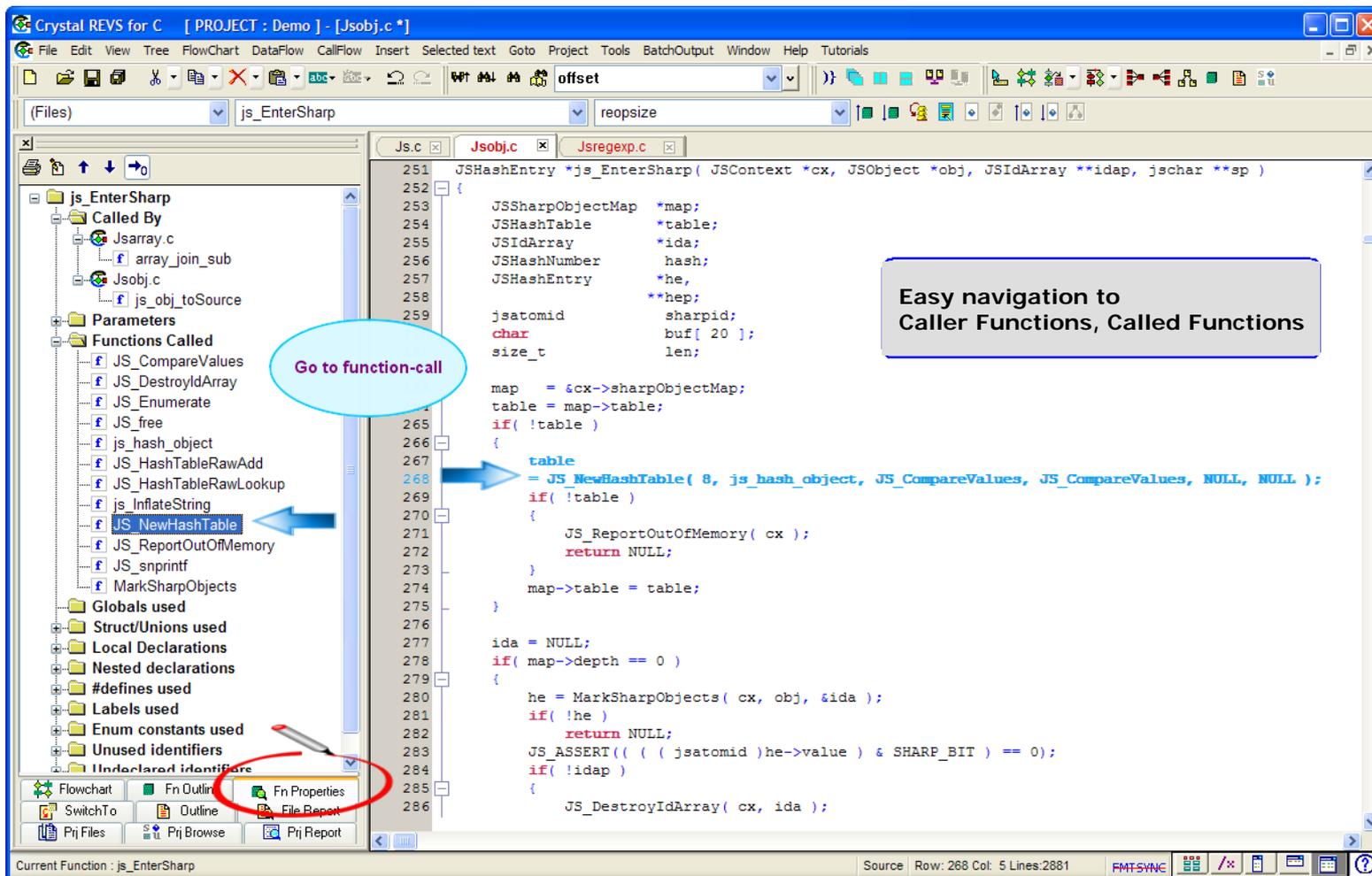
```
state->progLength -= reopsize
[ REOP_CCLASS ];
state->progLength += reopsize
[ REOP_UCCLASS ] + size;
```

The IDE interface also shows a project tree on the left and a bottom status bar indicating "Source Row:1689 Col: 1 Lines:3633".

You can see the control flow that affects the use of "reopsize".

You get a better understanding of how data is used - and you save time.

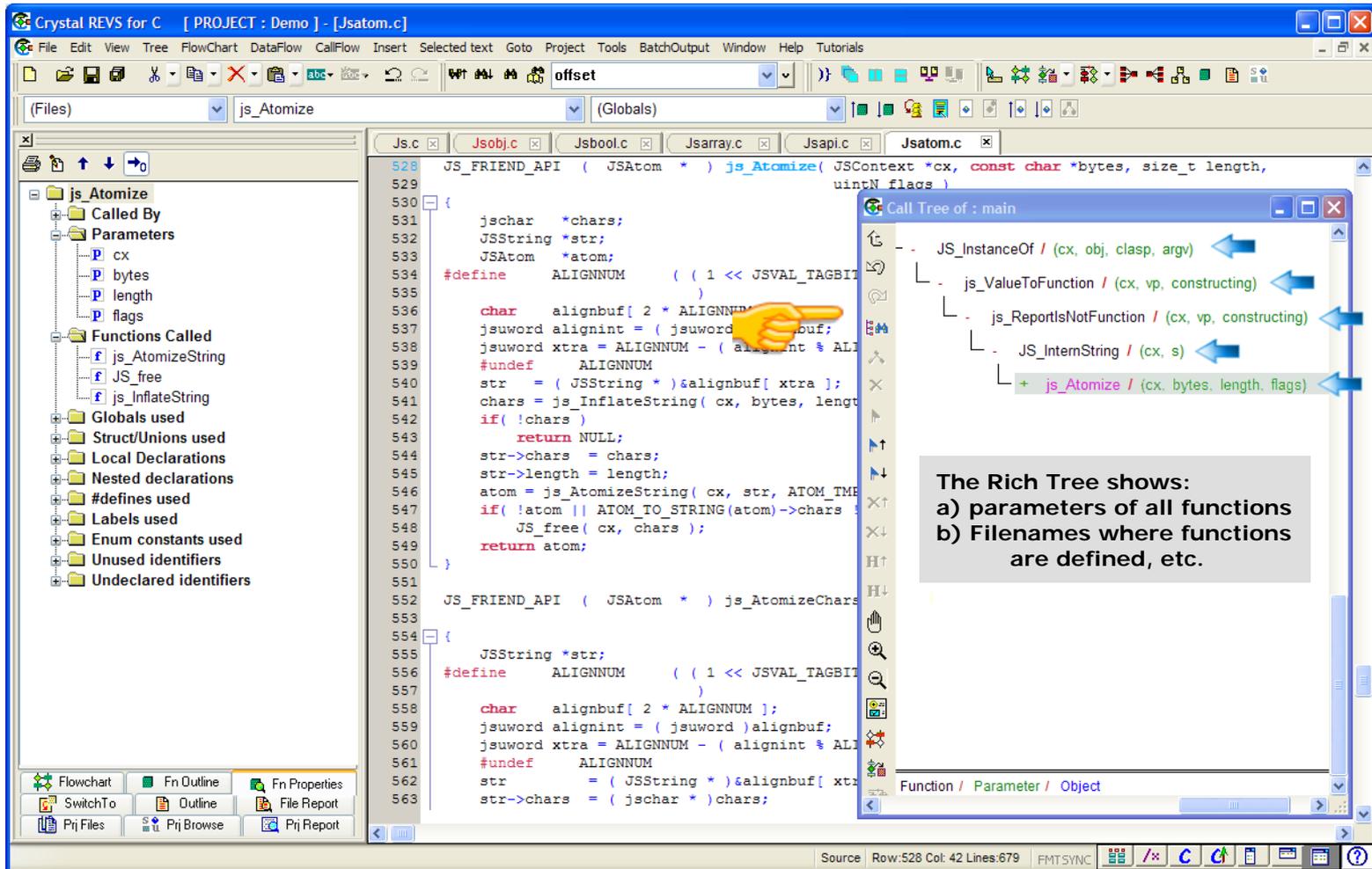
2. View the Function Properties card to view the Caller functions and Called functions



- With single-click, you can go to the function-calls in the code
- With a double-click, you can go to the function's definition

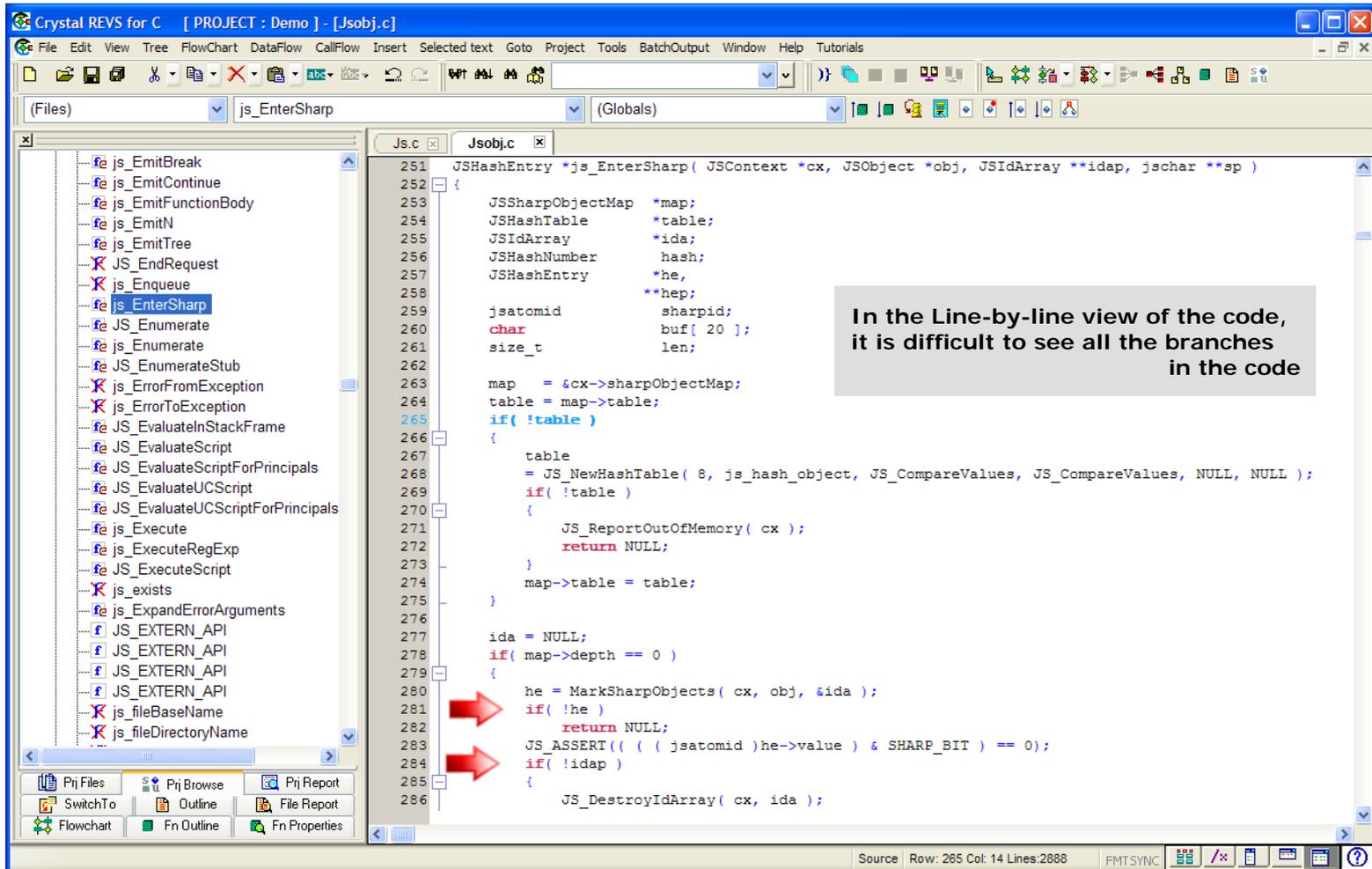
Use the Rich Tree:

View the parameters in the call-sequence



- You can view the parameters of all the functions in the call-sequence.
- You can view the filenames where the functions are defined

Check the modified code for correctness



The line-by-line view of code is not suitable for checking for correctness.

Here, it is difficult to see all the branches in the code. You can not see the whole function.

With Flowchart, it is easier to check the function for correctness

The screenshot shows the Crystal REVS for C IDE with a project named 'Demo' and a file named 'Jsobj.c'. The main window displays a 'Code plus Comment Flowchart of : js_EnterSharp'. The flowchart starts with a decision diamond 'if(map->depth == 0)'. If YES, it goes to a process box: 'he = MarkSharpObjects(cx, obj, &ida); if(!he) return NULL;'. If NO, it goes to another process box: 'hash = js_hash_object(obj); hep = JS_HashTableRowLookup(table, hash, obj); he = *hep;'. This is followed by a decision diamond 'if(!he)'. If YES, it goes to 'he = JS_HashTableRowAdd(table, hep, hash, obj, (void *) 0); CHECK if(!he)'. If NO, it goes to '*sp = NULL; sharpid = 0; goto out;'. The flowchart continues with 'JS_Assert((((jsatomid) he->value) & SHARP_BIT) == 0);', a decision diamond 'if(!idap)', and a process box 'JS_DestroyIdArray(cx, ida); ida = NULL;'. Finally, it reaches 'sharpid = (jsatomid) he->value;'. Annotations on the left side of the flowchart state: 'See all the branches in the code', 'Check the function in less time', 'It is easier to detect errors', and 'Reduce debug & maintenance costs'. The bottom status bar shows 'Source Row: 251 Col: 5 Lines:2881'.

- You have the whole view of the function
- It is easier to detect errors
- **The flowchart reduces debug and maintenance costs**

**In a Team Discussion,
Use Crystal Flow to make the discussion quick and productive**

Suppose you are in a team discussion to solve a problem.

1) You are trying to figure out **why a data object has an incorrect value**

Use the **best tool** for the problem:

View the Data-dependency Tree of that data object

2) You are trying to examine **all paths that can reach a given point in a function**

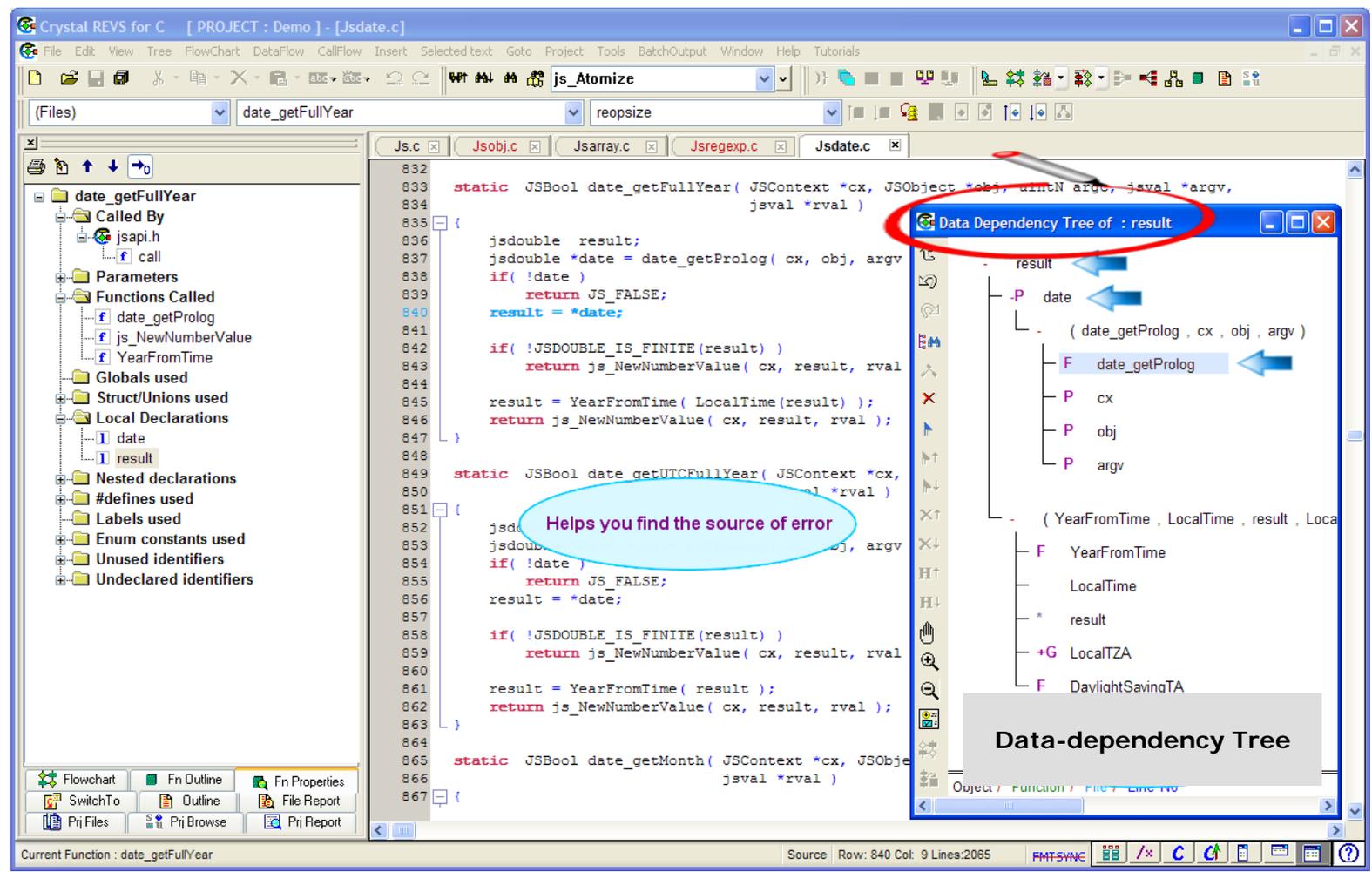
Use the **best tool** for the problem:

View the flowchart and highlight all paths to that point.

Question:

Which would you prefer for team discussion: **Graphical view** or **Line-by-line code?**

Data-dependency Tree: To find the source of incorrect data-value

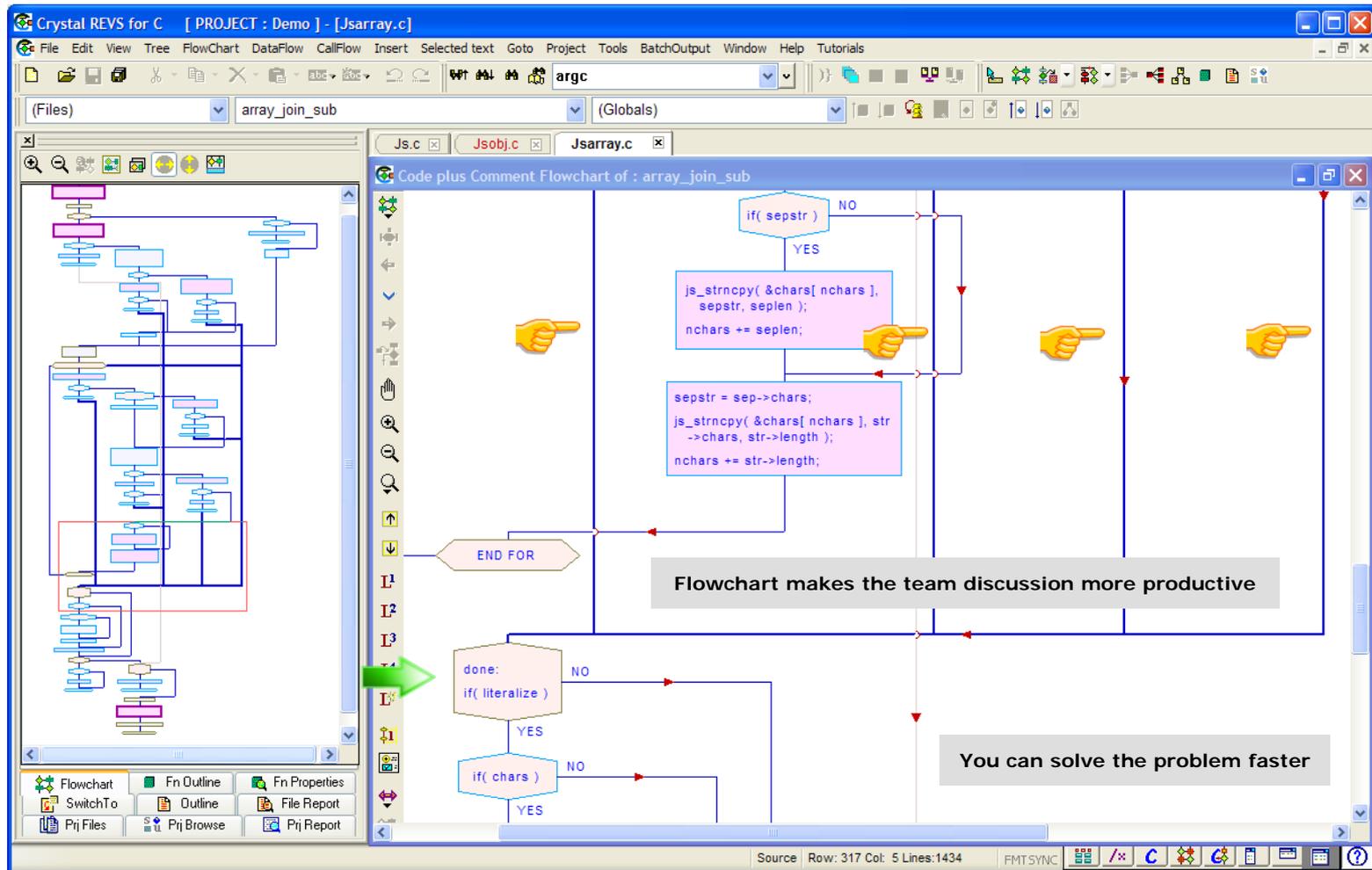


Here we see that the value of "result" **depends** on the value of "date" which, in turn depends on the function "data_getProlog()"

The Data-dependency tree helps you go to the source of error.

During the discussion

How do you view **all the paths** that reach a given point in the function



- Use the flowchart to see all the paths that can reach the point of interest
- Flowcharts make the team discussion more effective
- You can find the best solution in less time

Explore and Analyze Code in Less Time

- ◆ Do you need to **get an overview of a function** and **the functions called by it** ?
 - **Use CallFlow** - You can expand function-calls and go deeper in the CallFlow

- ◆ **Improve readability:**
 - Crystal performs code formatting. **Makes the code easy to read**

Use CallFlow to make a fast attack on a function and the functions called by it

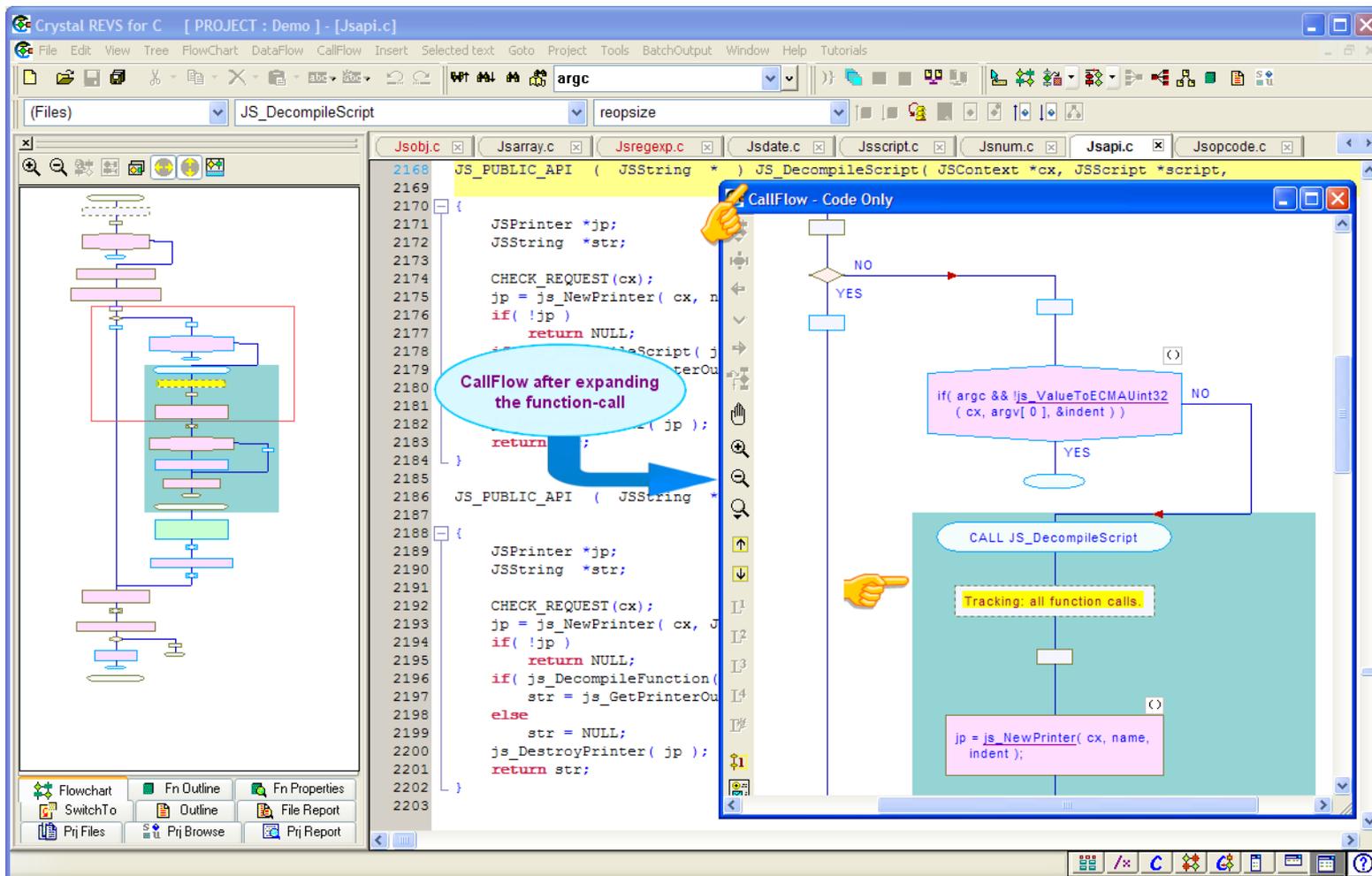
The screenshot displays the Crystal REVS for C interface. The central pane shows the C source code for the `script_toSource` function. The `CallFlow - Code Only` window is overlaid on the code, showing a control flow graph. The graph starts with a call to `JS_snprintf`, followed by a decision diamond. The 'NO' branch leads to a call to `JS_DecompileScript`, which is highlighted with a red box. The 'YES' branch leads to another call to `JS_snprintf`. A blue box labeled 'CallFlow of script_toSource' is positioned above the graph. A yellow box labeled 'You can expand the function-calls to go deeper' points to the `JS_DecompileScript` call. A light blue box on the left says 'Use CallFlow to make a fast attack on the functions'. The code editor shows the following code:

```
45 static JSBool script_toSource ( JSContext *cx, JSObject *obj, uintN argc, jval *argv,
46 {
47 {
48     JSScript *script;
49     size_t i,
50         j,
51         k,
52         n;
53     char buf[ 16 ];
54     jschar *s,
55           *t;
56     uint32 indent;
57     JSString *str;
58
59     if ( !JS_InstanceOf ( cx, obj
60         return JS_FALSE;
61     script = JS_GetPrivate ( cx
62
63     /* Let n count the source
64     j = JS_snprintf ( buf,
65     n = j + 2;
66     if ( !script )
67     {
68         /* Let k count the com
69         k = 0;
70         s = NULL;
71     }
72     else
73     {
74         indent = 0;
75         if ( argc && !js_ValueT
76             return JS_FALSE;
77         str = JS_DecompileScri
78         if ( !str )
79             return JS_FALSE;
80         str = js_QuoteString(
```

Here, you see the CallFlow of the function `script_toSource()`.

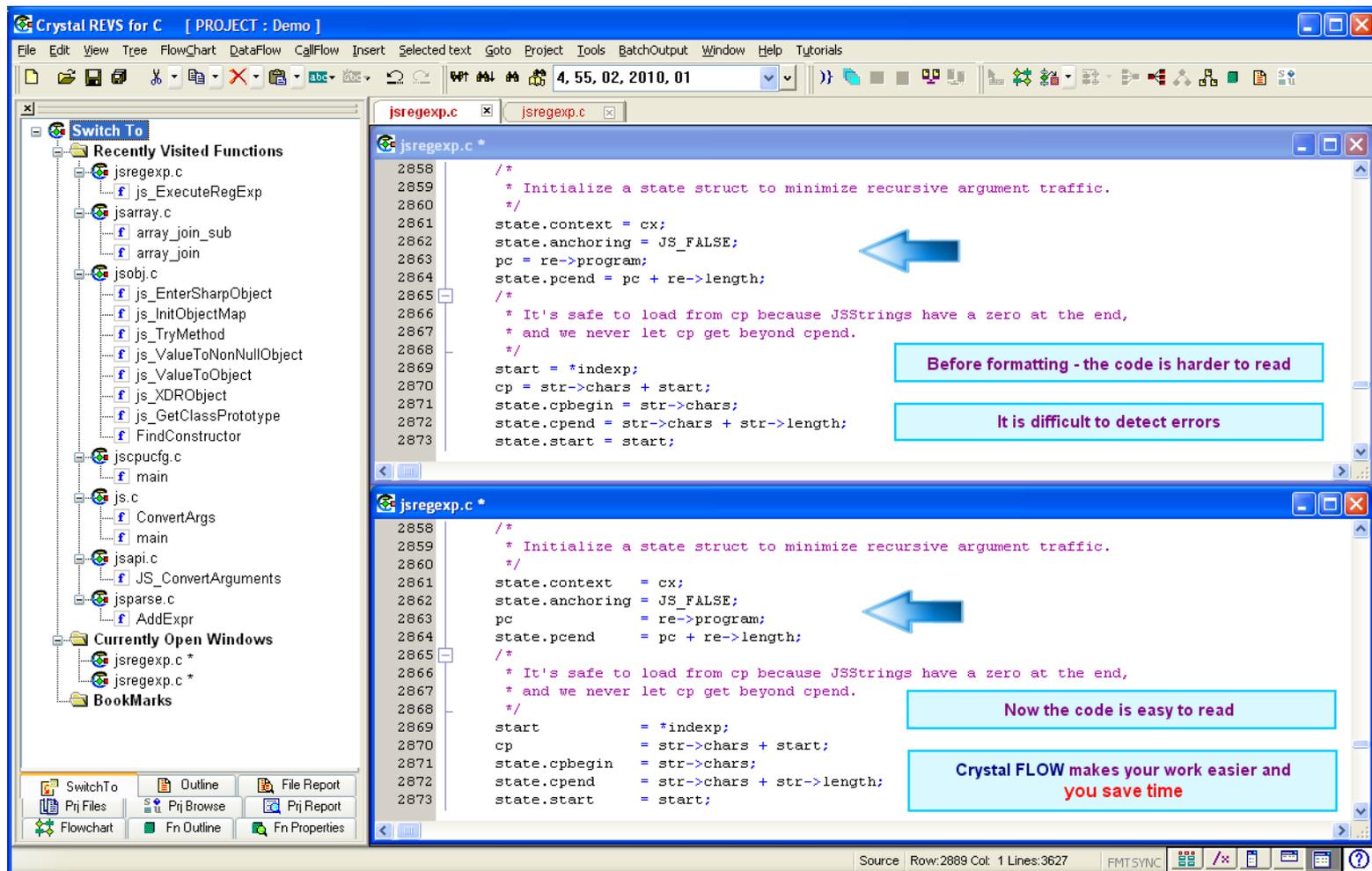
It shows the function-calls and the control flow around the function-calls.

Expand CallFlow to go deeper along the function-calls



- Navigate easily among a set of related function
- You can quickly analyze a function and the functions called by it

Crystal performs code formatting and **makes code and comments easy to read**



When you use Crystal, **you save time** at every step of your task.

Use the best tools.

Do a better quality design in less time.

Isn't it the formula to get ahead and stay ahead?

In the global marketplace,

If you lose 1 hour everyday, can you keep up with the competition?

Get Crystal FLOW.* *Be ahead of the competition.

With Crystal Flow, you can save more than **1 hour every day!**

Use Crystal Flow. Increase individual productivity. Increase team productivity...