

Context

Steven Teleki

Most programmers know about "*context switch*" if their education or experience included at least some amount of assembly level work. In the "context" of assembly programming context refers to the contents of all CPU registers.

Briefly, when the operating system decides that a process ran enough, then it executes a context switch, and lets some other process run for a while. During the context switch all relevant CPU register data is saved so the process that had to give up the CPU can come back and continue work as if nothing has happened.

I believe that we can apply context to the larger issues of software development work.

What is Included in the Context?

Context for me is everything that I need and use to write software. This includes:

- ❑ **Requirements.** Either written or verbal, most of the time verbal and vague. The written requirements are sometimes formal, other times are hastily written notes on little pieces of paper. Yet other times are emails or email fragments.
- ❑ **Designs.** These can be in form of plain text or box drawings on a whiteboard; sometimes UML-like diagrams. At times the design is written down on paper, in Visio files, or in some design tool.
- ❑ **Prototypes.** They are sometimes hastily drawn user interface mock-ups. Or they are code prototypes that prove that an idea can be implemented. Or they prove that two components can talk to each other with the envisioned interface.
- ❑ **Existing Code.** There are many times that you have lots of existing code. Managing it and figuring out exactly what to change and how to add new code to it can be very challenging.
- ❑ **Plans.** They are somewhat an odd lot in this list. Plans are the result of synthesis of the information already on hand. As a result plans can be hard to create, because synthesis is hard. They can be hard to maintain, because plans have to evolve as you learn more about the project.

As you can see, context can show up in many different ways. And on some projects you might have all of them. Managing them is not easy task, so no wonder, that organizing your context can be a daunting task.

Why Build Context?

Generally context guides you in the direction of what to do next. Here is a short list of the reasons:

- ❑ Context helps you when you need to recover from short interruptions (phone, ICQ) or long ones (end of the day, other project getting temporary high priority).
- ❑ Context guides you when you need to do add-on work to an existing project (or guides others if they have to do it).
- ❑ Context helps others (or you) to (re)validate your thinking against both new and old information.

Do you agree that when you start work in the morning it is almost always a struggle to build up the picture that you held in your head the day before? Does it take you a half an hour just to write the first line of code? This time is spent on what I call the building of context. If you have some *crutches* that help you, then you can build up the context of the work in your head faster, and you can be productive faster.

How Much Context Do You Need?

Just the right amount. Striking a *delicate balance* between too much and too little context is challenging and may only come with experience. I would say that if you have trouble picking up your work after an interruption, then you should look into adding more context than what you currently have.¹

My rule has been that if it takes me more than 10-15 minutes to becoming productive after a 10-15 minute interruption or at the beginning of the day, then I don't have enough context and I need to think of ways to create more.

Some developers believe that *the context is in the code*. Well, it depends on the code. I believe that it is much easier to find your

¹ Conversely, if you notice that your work is not getting done then check to see that you are not spending all your time creating context.

way around the code if you have a few "maps" or at least some "signposts" to guide you. I do believe that it is possible to build context into the code, but I have rarely seen it done well.

The Power of Context

In *The Tipping Point*, Malcolm Gladwell talks about the Power of Context and he asserts that by tinkering with the smallest of the details of your environment (your context) you can achieve extraordinary results. When you apply this powerful idea to software development you can find new ways of looking at some very familiar things.

Here is an example that I have seen repeatedly: you develop software and you have a fair number of defects in system test. You don't design before you code, except maybe an occasional quick diagram on the whiteboard. Try creating a reviewable design, review it, then code, and see if you can cut down on the number of defects that you have to find and fix in system test.

How Can You Put Context Into Your Code?

Here is a way that you can build context into your code, or rather, organize the code itself to provide the right context: Organize your code into components of about 200 to 2,000 lines of code and have clearly defined requirements for each

component. Consider each component a separate project. This will make it possible to comprehend a component with little extra help.

Notice that I didn't say that you must have the requirements *before* the work starts, but you should have the requirements finished when the component is finished. Consider the requirements for a component a valuable documentation that helps the people coming after you to understand—quickly—what you did.

What do you think? How do you cope with the *context* problem? Send me a note and let me know.

Steven Teleki is a software development manager, coach, and mentor. He has 16 years of wide-ranging experience in the software industry. He is passionate about creating high-quality software on time and on budget. He focuses on understanding and improving both individual and team software development performance. Contact him at teleki@acm.org.