Programming Stories in English

In his superb memoir, *On Writing*, Stephen King says that rather than deciding what he wants in a story and then writing to make it turn out that way, he puts characters in interesting and difficult situations, then writes to find out what happens. He says stories are "found things, like fossils in the ground", and says he plots his as infrequently as possible because life is largely plotless despite all our precautions and planning: "Plotting and the spontaneity of creation aren't compatible. … [M]y basic belief about the making of stories is that they pretty much make themselves. The job of a writer is to give them a place to grow."

In a 2008 interview about the twelve brilliant *Fawlty Towers* TV farces he and Connie Booth wrote, John Cleese said that they never really started to write the dialogue until they'd got the plot worked out in considerable detail. They might get stuck for a while. And they always had to change it *a bit* if they found that a scene didn't work. But this was key: They never started until they had the story line, so they always knew where they were going. He said: "Some people try to write comedy by starting, Scene 1, and then start writing the dialogue. Well, the chances of them getting to a satisfactory ending are one in a hundred. You've gotta kinda know where you're going while you're building the thing."

Both King and Cleese are absolutely right. But most authors are hybrids somewhere between those "seat of the pants" and "plotter" extremes. And even King, Cleese, and Booth do both: King the pantser does his plotting during the edits after seeing the fossil emerge. He digs as a plotter to bring out the story he has discovered. Plotters such as Cleese and Booth do a lot of plot pantsing while following their idea, but they do most pantsing later writing the dialogue.

A story is a series of happenings that resolve into an ending. Each author gets there by some combination of pantsing and plotting. This article is not biased toward either extreme. Rather, it embraces both as being necessary functions no matter which kind of writer we think we are. But any of us can get stuck with our pants down, or stuck with our plot circling the drain.

My first move to get past these problems is to sidestep any sticky place by putting a signpost *in the text* to confess the problem, like: **[Here, yyy needs to happen so xxx can happen later.]** Sometimes I know what yyy and xxx are; sometimes I don't. In the example below, I didn't, and its signpost could read: **[As a break, put a scene here of something else that needs to happen.]**

Honest confession admits failure. We may not like to, but it's crucial: We can then stop berating or defending ourselves, and start working on the problem subconsciously. This frees us to move forward consciously. Computer programmers call that background thought process a *daemon*. It looks for certain sets of conditions, then signals the main program whenever it sees them.

Sometimes the daemon—authors may call it a muse—wakes us up at 3 a.m. with a story bone to replace a confessing signpost. But if not, eventually time runs out. So, it's best to have some technique for turning problems and gaps into signposts or bullet points, and then into scenes. Techniques save time—the time we would spend drinking and moaning about being stuck.

The essence of "getting stuck" is either that the characters (including the narrator) stop moving around and talking in an author's head, or that their movements and talk seem uninteresting. The only way to get un-stuck is to get them going and talking well again, and the best way to do that is **to start peppering our muse with questions**. Let's look at the questions and history of a short scene to illustrate the process I use for turning story bullet points and signpost notes, step by step, into an edited but rough draft. Maybe it will help others do the same.

The plot of a story is the series of things that happen. Some people use the word "outline", but I prefer Stephen King's metaphor of a spine: It is a series of bullet point bones arranged on a story board or in a stack of 3x5 cards. As a pantser, I write them down as I get them. As a plotter, I re-arrange them. I may find other bones to hang from the spine as if they were arms, legs, fingers or toes: *i.e.*, bullet sequences of subplots, setups, payoffs, and so on. Greg Sager, a friend with a Masters in theater arts from Kent State University, used this spine system. I stole it from him (and Walt Disney) such a long time ago that the Statute of Limitations has expired.

SPINE BONES EXAMPLE

Two bickering engineers are driving around, trying to hunt down a Thinking Machine (*i.e.*, TM) that is acting oddly. (It doesn't have GPS for reasons I won't bother to tell you.) The engineers reveal themselves plus other things as they argue along the way. In the spine, this hunting sequence started as a bunch of bullet-point bones and a few bits of dialogue which all must happen (the bickering; the spy that was two spies; the intern; the gossip about Bobby; *etc.*). One by one I turned those bullets into edited story text using my process. So far, so good.

Although everything to this point was either setup, payoff, or other useful stuff, I felt readers needed a break—a brief change of location and character. Any new scene must advance plot and be interesting for sure, and maybe reveal character. But it also must: "Show; don't tell."

But show *what*? What *needs* to happen soon in the story which isn't part of the scene I'm interrupting? I asked my muse that and other questions, and here is the result, shown in blue. It's pure "tell" (getting some "show" from the muse would be a bonus). The "." Line below it is a *stub*. That's programmer jargon. In programs, stubs are do-nothing code with a meaningful label, which will all be replaced later by *subroutines* or *method calls* or whatever. In my English sci-fi story, it will be replaced by more *story*. (Any yyy xxx or other signpost is a kind of stub.)

- **The troubles trigger complaints from Chairman Bozworth, who says the Board has heard about odd machine behavior. He wants answers from Doc.**

  "."

METHOD and MUSE

Since this new scene must advance the plot and possibly reveal character, what's the next step? **Make a list of *input* things taken from the signpost bullet**. The things are: any ideas, key words, and actions which the scene needs in order to work. Writing this list into the stub might bring up things to add that are not in the bullet. If so, I'm happy: The story is talking to me again.

Next, I must write the questions Boz would ask. Note that I do not try to list Doc's answers yet. If the muse blurts them, I'll write them down, but I'd rather they came fresh out of the faucet after I write each of Boz's questions *in the story text*. Finally, I must **list any output things that can or should happen as results *inside the scene***. Thus, in red and brackets, we get:

- **The troubles trigger complaints from Chairman Bozworth, who says the Board has heard about odd machine behavior. He wants answers from Doc. [Boz asks: What is odd? Which reseller? Any customer complaints? What are you doing about it? In return, Doc wants to know how Boz knows. (Doc is trying to identify leaks.)]**

  "."

MORE PANTSING

In this step, I put those listed things into the stub as story text, and I should probably include the scene intro, which should use Doc's POV. But I had nothing specific until I stared at the bullet while imagining Doc's office with Boz coming in differently than he does in other scenes. If I had stayed stuck, I would have imagined them meeting in some other place because it doesn't matter where they meet. But I got a picture of Bozworth carefully shutting Doctor Little's office door. Usually, Boz is energetic. So this change of pace immediately grabbed me, and I was able to pants the scene from there. I didn't worry about "show versus tell" or other editor-hat things like removing adverbs, and so on. Note that I left out the "Which reseller" input after a story muse whispered to me that although it is necessary, it's rather obvious. Immediately a muse blurted that its absence could therefore serve as a setup clue for later:

- **The troubles trigger complaints from Chairman Bozworth, who says the Board has heard about odd machine behavior. He wants answers from Doc. [Boz asks: What is odd? ~~Which reseller?~~ Any customer complaints? What are you doing about it? In return, Doc wants to know how Boz knows. (Doc is trying to identify leaks.)]**

Winston closed the office door slowly and quietly. It was clear to Dr. Little that the chairman was gathering his thoughts.

"The Board has heard about odd problems, Maynard. What is this odd business?"

"We are not sure. It may be serious or silly, and we're trying to find out which. How did you find out?"

"What are you doing about it?"

"I would rather not say just now."

"Have you had any customer complaints?"

"."

KEEP GOING

I am a hybrid writer somewhere near the middle between Pantser and Plotter. Pantsers will surely get more scene text than I got once they start writing. From my programming roots, though, I like to know approximately where I'm going before I start, which makes me more a Plotter, I guess. But once I start on a scene, a chapter, or whatever, I *try* to pants it to some interesting end and leave artsy presentational features until later (unless some manage to spurt out of the faucet now). If a new bullet point pops up, I'll add it or put it in brackets for later.

Occasionally the pantsing step is interrupted by an unrelated idea or picture, or by characters that argue or discuss something. True pantsers ignore it and push on, but I prefer making any valuable stuff into a bracketed signpost right where I am in the text. I can relocate it later.

Here is an intermediate version of the Doc/Boz scene. It might have gaps that I will possibly recognize and fill with more keys and story. I edit *a bit* as I go because when I programmed in Java, the IDE insisted on good syntax,[1] and old habits die hard. Also, this editing-as-we-go is what some programmers call "blacksmithing" which beats the program into shape, making it do what we want in stages much as the cycle of re-heating and beating will forge metal into shape. When writing stories, I compare it to bouncing between plotting and pantsing. Please note that Boz's comment, "The Board has heard", is now *reversed* to enhance suspense. Plus, I added things (outputs) in a new bullet *after* the stub **to clarify what the scene should produce:**

- **The troubles trigger complaints from Chairman Bozworth, who says the Board has not heard about odd machine behavior yet. He wants answers from Doc. [Boz asks: What is odd? ~~Which reseller?~~ Any customer complaints? What are you doing about it? In return, Doc wants to know how Boz knows. (Doc is trying to identify leaks.)]**

Winston Bozworth thanked Marie and stared at the office door as he closed it slowly. It was clear to Dr. Little that the chairman was assembling his thoughts and they must be important.

"The Board has **not** heard," Winston said after taking the best chair in the same thoughtful ease. "But there will be problems when it does. I assume you can guess what I'm talking about?"

"It would be odd if I didn't."

Winston smiled and (surprise) asked for Kona coffee. As Doc puttered, the chairman pitched questions.

"What about this odd TM business, Maynard?"

"We are not sure. It may be serious or silly, and we're trying to find out which. How did you find out?"

"I have to say this is a serious problem and we need data before the Board finds out. What are you doing about it?"

"I would rather not say just now."

"Have you had any customer complaints?"

"No. Have you?"

"No, thank God."

"."

- **[Doc gets no clues about the leak *except* that Boz didn't ask which TM reseller has the odd machine. (Does Boz know which one it is?) Doc stonewalls Boz about the odd behavior. Boz lists possible damages (foreshadowing).]**

ROUGH DRAFT

Here is the edited rough draft with more key words highlighted. The additions are not crucial enough to bloat the signpost list, but they foreshadow important things, and it is a good idea to

[1] IDE stands for Integrated Development Environment. At a minimum, it helps programmers use correct syntax and calling sequences to speed the coding process and avoid bugs that can otherwise be hard to find. Authors have spelling and grammar checkers, but a programmer's IDE is never flaky like an author's tools can be.

keep track of them by using color or a word you can search for easily in case the story changes. The stub is gone; the scene is complete, but rough, and I'll smooth it out later when I'm editing (*i.e.*, debugging). Also, "the stock price" is so important that I should have included it earlier. …And some artsy presentational spurts made it into the first paragraph:

- **The troubles trigger complaints from Chairman Bozworth, who says the Board has not heard about odd machine behavior yet. He wants answers from Doc. [Boz asks: What is odd? ~~Which reseller?~~ Any customer complaints? They affect the stock price. What are you doing about it? In return, Doc wants to know how Boz knows. (Doc is trying to identify leaks.)]**

Winston Bozworth thanked Marie and stared at the office door as he eased it shut. Dr. Little sat straighter, certain that the chairman was assembling his thoughts and they had sharp edges.

"The Board has **not** heard," Winston said after taking the best chair with that same thoughtful ease. "But there will be problems when it does. I assume you can guess what I'm talking about?"

"It would be odd if I didn't."

Winston smiled and (surprise) asked for Kona coffee. As Doc puttered in his tiny commissary, the chairman pitched questions.

"What about this odd TM business, Maynard?"

"We are not sure. It may be serious or silly, and we're trying to discover which. How did you find out?"

"We have to say this is a serious problem and I need data before the Board finds out. What are you doing about it?"

"I would rather not say just now."

"Have you had any customer complaints?"

"No. Have you?"

"No, thank God."

Accepting a steaming mug, Winston added, "Are you going to consult Legal?"

"Certainly not. What are you afraid of? This is probably a configuration issue."

"I'm not so sure."

Doc's sharp "Why?" cut through the polite calm.

After a hitch, Bozworth said, "I'm more concerned with the marketplace than you are."

Little sighed, "That's as it should be, I suppose."

After more sparing and some small talk, the chairman asked, "Is there anything Marketing should know—about changes to the existing machines?"

Doc laughed.

"Okay, Maynard, have it your way, but such things affect the stock price."

Doc gave him the Eyebrow Look, and they shared friendly banter as Winston left.

Then he reached for the phone.

- **[Doc gets no clues about the leak *except* that Boz didn't ask which TM reseller has the odd machine. (Does Boz know which one it is?) Doc stonewalls Boz about the odd behavior. Boz lists possible damages (foreshadowing).]**

ANNOTATIONS

Programmers should pepper their programs with Remarks, *i.e.*, plain-text notes to themselves (and other programmers) about what the code is doing. Notes are identified by a symbol (*e.g.*,

// as in Java) or a word (*e.g.*, REM as in Basic) or by whatever the programming language uses. They aren't compiled into the executable machine code, but such notes are necessary for programmers because Maintenance of existing code is the most common programming task.

It's the same for authors who program in English: *Editing* is the most common task. My bulleted paragraphs and bracketed notes are like Remarks I made to myself. Once a story draft is fit to send out, I'll save it in an archive directory. Then I'll copy it, strip out all bullets and notes from the copy (I've made them all easy to find), and send it off with any necessary cover letter.

SUMMARY

The input and output lists are key for making this process work to both show and tell. Plotters and Outliners tend to get the inputs and outputs before they "pants" the scene. Pantsers tend to write the scene and—if they are wise—will understand the scene well enough to highlight any inputs and outputs they can use for other scenes. (Unused outputs are prime candidates for deletion later but may also be inspirations for sub-plots or sequels.)

Computer programs and novels have many modules which all need inputs to produce outputs.

- Programs have: commands, subroutines, methods, applications, systems, *etc.*
- Novels have: sentences, paragraphs, scenes, chapters, sequels, and franchises.

Any module must produce outputs from the inputs so that later modules have what they need.

- *Programmers then run their code,* see bugs and other problems, and rewrite to fix them.
- *Authors read their story*, see typos, POV, and other problems, and rewrite to fix them.

Do you remember what I said about computer program modules called *daemons* which do things the application's user never sees? Can a writer's stories have them?

Yes, and most stories do. They are called *backstory*. They are rarely seen (as scenes), but have a great deal to do with what happens. One very important benefit of my bulleting method is that bullets need not be turned into scenes. They can sit there *at the proper place in the timeline*, reminding the author of things that may even determine how the story ends.

For that reason, I won't bash authors who put backstory at the beginning. Instead, I'd advise them to leave it there, bulleted or bracketed in the master document as shown in my example above, and then re-think how to show its important effects throughout the story text. Then, of course, strip it plus all other bullets and notes out of the story text prior to submission.

This technique also helps us get un-stuck or un-blocked when writing a *bulleted scene* (after we've unblocked ourselves in the *plot* by using bullets to get past sticky places). For example: The scene above is from a novel I'm working on. A couple of times (so far), the necessary events in a bullet or sequence of bullets were either difficult to write or—if written as scenes—would have slowed the story down. So, I kept those bullets in place to remind me what's happening. Then I included just their important points in other scenes but as backstory, setup, or payoff.

There are other *curiously different* similarities between programming and novel writing:

Programmers have a dictionary (an instruction set) where word meanings never change with fashion, and they use its words to tell machines what to look for and what to do. However: such program *content* (*i.e.*, commands and data) has no *meaning* for the machines themselves to grasp or misinterpret (unless the machine is said to be intelligent—and we'll debate that later).

Authors have dictionaries which change a bit every year, and words in them carry meaning as well as content to the biological machines we call "readers".

This distinction between *content* and *meaning* applies to the words *tell* and *show*. The stock advice to authors is: Don't tell it, show it. But what is "it"? …I offer a programmer's perspective:

Writing, like life, is loaded with paradox.

All writing is "telling". Even in that scene which shows Bozworth assembling his thoughts, I *tell* readers how he showed it. To reinforce the show, I tell Dr. Little's "sat straighter" reaction. "Showing" is what Shows do: Our television, movies, and stage plays show more things better than writing can, and the most effective shows are the ones we give each other every day. Our *written tells* may come close to showing—sometimes very close—but the phrase "you had to be there" hints at the chasm between them.

However.

All writing is "showing". With every word, punctuation mark, and even blank space, readers *automatically* build or recall mental constructs of feelings, images, ideas, events, even other words. In fact, producing those mental constructs is the entire function of words and language.

So, it turns out that *tell* and *show* are the opposite extremes of doing the same thing. This gives us a clue that each might be better than the other for some things, and worse than the other for other things. For example: Show is slow and generally takes more words to produce the mental constructs, but they are usually richer. This richness may bog the story down. But if the richness is too brief or colloquial, it may produce wildly different mental constructs in different readers, so, in places where a story needs narrative drive or clarity, prefer a telling word like "shouted" or "sighed" ("Yes," he sighed, "I do sometimes sigh while I say words."). Of course, some telling words like "say" and "reply" are practically invisible; they carry no color or texture and produce nothing in readers except the idea that other spoken noises carry a meaning.

But no matter whether we use *show* or *tell*, it must be *interesting*. If a character or setting needs to be rich, then give readers a feast but omit useless words. When narrative drive is important, take the time to find words that quickly suggest the right color or texture. And unless a character likes to use clichés, try to avoid them—unless they are thematic or give a twist somehow.

This article and my bullet/signpost list technique are mostly about how to create or discover good telling bullet points and turn them into showing scenes. But my emphasis has been on the *creating* and the *turning*, not the showing. I have focused on input/output things themselves,

rather than on how to render them, because there are boatloads of excellent advice already on how to show.

I *told* you **the most important key** once, but *showed* evidence of it a bunch of times. Whether pantsing, plotting, finishing, editing, or whatever the problem is, ask your muse questions like:

- What other *interesting* thing could go wrong with his plan?
- Why would she do that?
- Why isn't this scene working? Is it necessary? Why?
- Would this line of dialogue be better coming from a different character?
- …And so on.

I've suggested all along that writing stories is much like programming computers. So, did I talk to my muse when I wrote code for Goodyear, Sherwin Williams, American Greetings, *et al*? Certainly! But when I worked for businesses, my boss first gave me a full specification or at least a bullet list of everything the code had to do. Authors in English must make up those inputs on their own (like Booth and Cleese), or plug along hoping they appear (like King). Could one reason why authors get stuck be that they focus on how to say something, before they know well enough what they want to say? That is the problem my method solves—for me, anyway. Maybe it will for you.