Alex Groce (agroce@gmail.com), Northern Arizona University

Gerald M. Weinberg's *An Introduction to General Systems Thinking* is this month's Passages selection. Weinberg's *Psychology of Computer Programming* was last month's Passages selection. I believe that many more software engineers have read last month's selection than this month's. Some of the virtues of *Psychology* are shared by this book: Weinberg's engaging writing style, cultural and literary fluency, clever questions for further research, intellectually exciting suggested readings, and overall combination of a big-picture theoretical framework that is suggestive and powerful with a practical, anecdotal approach that makes it possible to get down to brass tacks. However, the two books are also very different: unsurprisingly, the *Psychology* is clearly a very relevant book for software engineers, about the human dimension of programming computers. *An Introduction to General Systems Thinking* uses computer programming, as an example of a way to go about doing general systems thinking, but it is not really a "computer" book, specifically, at all. It is what it says it is.

Why, then, is it a book specifically for software engineers? To understand why, you'll need to let me (seem to) digress for a bit, by repeating verbatim (a part of) some notes I took while re-reading Weinberg's *Introduction*:

> *I can, with existing code, grep for a TERM from a real domain, but that term is, usually, arbitrary language (it could be "foo" and system would still work same) - NOT programming language constructs, or machine elements, but at best defined by a library or other system we can't modify. But usually OURS*
>
> *Our programs usually map from some context to an operative model; makes executions like science makes predictions*
>
> *Our effort is much closer to the scientific enterprise, or inventing an engineering discipline, than to "normal" engineering in a well-defined discourse/domain. We have to "Kuhn it up" pretty often if doing "real work" vs "just grunt coding". I personally find just grunt coding useful and somewhat fun at times, but what a way to make a living!*
>
> *Underlying idea of "passages" is that interesting SE is like doing science. Hence Weinberg's book is useful.*

What? I am claiming that software engineering (as opposed to "mere coding") often involves, at heart, constructing a model of some aspect of reality (either real-world or computational) that is provisional, complex, and yes, a little bit *ad hoc*. It's experimental, in the sense that you seldom fully understand the thing your software talks to, or represents, and must "feel" your way to an approximation of its behavior.

I hypothesize that if you are good at experimental science, in domains with notable uncertainty and lack of foundations, but that are not completely "loosey goosey" (e.g., not critical theory or astrology), then you should, all things being equal, be good at software engineering, too.

Weinberg's book is about how to go about doing "experimental science" in domains where getting everything right is too hard, and aggregate methods don't apply well -- Weinberg correlates this kind of work to a "law of middle numbers":

> *"For medium number systems, we can expect that large fluctuations, irregularities, and discrepancy with any theory will occur more or less regularly."*

Things in the realm of "middle numbers" lie between the world of "small numbers" where you can just compute everything because the number of interactions is small, and the world of "large numbers" where statistical properties dominate, and averages are very useful in prediction. In between, it gets hard. It gets muddled. It gets, in short, like most serious software work. I feel comfortable thinking about this world, because software engineering research exists in precisely such a part of actual science: software systems are generally too complex to understand by reduction to a small number of parts, but do not behave like ideal gases, undifferentiated statistically comprehensible substances. Weinberg is addressing a problem that is more general than that of software, but a problem that definitely includes software, and the software sciences.

So what does Weinberg tell us about how to operate in this world? He offers the notion of General Systems Thinking, and associated General Systems Principles, as a tool to help us out. Note that one such principle is *"If you cannot think of three ways of abusing a tool, you do not understand how to use it,"* according to Weinberg, so don't be surprised if using General Systems Thinking is not a mere matter of memorizing and applying a few Thought Patterns.

There are only seven chapters in this short book, and the last chapter, rather than concluding anything, reads like a list of open questions, provoking arguments, and a launching pad for another book, one that Weinberg (to my knowledge) never really wrote. Before he gets to "THE END??!!??!", however, Weinberg offers up some key advice as to how to go about working with his "general systems." The basic principles offered up include composition, decomposition, abstraction, concretization, and black and white box methods: all things the software engineer should know, does know, but perhaps has not thought about in this context, before, or seen in this more general light. There is much to be said for standing back from a familiar thing, and seeing it as new, and unfamiliar, and a bit disconcerting. *That* principle is longstanding, and older than software; Chesterton knew it, when he wrote *Manalive*, and Eliot knew it when he said that "the end of all our exploring/Will be to arrive where we started/And know the place for the first time."

The principles of general systems thinking are, of necessity, contradictory; or, rather, they offer up, like a type system, principles that introduce and eliminate certain "wrappers" around "the thing itself" and so let us handle it according to our varying purposes, abilities, and mere inclinations: compose when needed, then decompose. Though the concept here is deeper than simply identifying two useful sides of a coin; there really is a contradiction, and even a war (with no desired winner) between the reductionist and the holist, the materialist and the idealist, the rationalist and the existentialist. Note that Weinberg actually refers to composition and decomposition not only as principles or techniques, but as *errors*, and this is another new way of seeing, perhaps the most important of all.

An Introduction to General Systems Thinking fits into what I consider a “line of thought” that is essential to building, understanding, testing, debugging, and maintaining complex software systems, perhaps best explained in Herb Simon’s *The Sciences of the Artificial* (it’s no surprise that Simon’s book appears in the reading list for Weinberg’s penultimate chapter). Namely, we must understand that we are very limited in our abilities, in some respects, and can easily build systems far beyond our boundedly rational capacity to plan and understand. Only by acknowledging that limitation, and working around it, can we get anywhere but into a muddle. This book offers some deep tools for working in the mud.