

Obstacles to Happy Endings

Colin Potts

College of Computing, Georgia Tech

(<http://www.cc.gatech.edu/gvu/people/Faculty/Colin.Potts.html>)

Requirements completeness is the extent to which the stated requirements for a system meet the user's needs. Because intent is subjective and changing, requirements completeness can be approached but never achieved. However, approach it we should.

Requirements incompleteness often manifests itself in the unpredicted inflexibility that a new computer-based system introduces into an enterprise. Users are often unpleasantly surprised not so much by what the new system does as by what it forces them to do to adapt to its inflexibility. Meeting schedulers are a perfect example. Depending on whether you are a rationalistic nerd or a sensitive humanist, meeting schedulers either operate in organizationally imperfect environments or situated contexts of great subtlety: People forget to update their agendas, or fail to respond to scheduling requests; they change their plans; rooms are closed for plumbing repairs. A meeting scheduler that was designed by people who refused to accept these facts would be incomplete.

We can only address the problem of requirements incompleteness in general by integrating methods from two design traditions: From the rationalistic tradition, we must refine further the goal-based analysis approach, so that it accounts for the many things that can go wrong in real processes; and from the ecological or ethnographic tradition, we must systematize the processes of observing workplaces and conducting workplace interviews in such a way that we can more easily translate our analyses of current processes and practices into prescriptions for intervention, support, and automation.

My poster describes an approach to goal refinement that gives central importance to *obstacles*. Obstacles are the conditions under which achievement goals may be blocked. We want to be able to explore such obstacles, and the way that the proposed system and its users will defend against obstacles and mitigate their effects, by walking through representative, or *salient* scenarios in which different obstacles arise. That is, we want to look at every significant obstacle and combination of obstacles, but we cannot be exhaustive. The idea of salience is therefore analogous to that of coverage in program testing, and I am currently investigating scenario generation heuristics that use criteria resembling coverage metrics.

Scenarios (if they are truly salient) are like stories: They have a plot, and the plot has a point. I have adapted story grammars, which were proposed in the 1970s to explain human memory for stories, to the generation of scenarios. My scenarios have episodic structures derived from a goal-based analysis of the system, and their plots turn on the occurrence of one or more focal obstacles. The scenarios can be represented textually or in tabular/diagrammatic forms. Judged as stories most of them are unexciting. But judged as design tools they are very interesting: How many of the scenarios have happy endings depends on how completely the requirements have been specified and therefore how flexible the system will be to non-normative user behavior.