PLDI: G: Programming and Debugging Surveys

Emma Tosch
School of Computer Science
University of Massachusetts
etosch@cs.umass.edu
Advisor: Emery Berger

Abstract
Surveys can be viewed as programs that may have bugs. These programs ought to be written in a language that encodes the appropriate abstractions and best practices of the survey literature. SurveyMan is a DSL and runtime system for designing and debugging web surveys.

1. Introduction
Over the last decade, web-based surveys and crowdsourcing platforms have become increasingly popular, making it possible to reach large and diverse populations at low cost [1][2][18][19]. This low cost, combined with easy-to-use survey design tools such as Qualtrics and SurveyMonkey, allows researchers and marketers to collect and analyze results faster than ever before.

Although these resources have democratized the collection of survey data, they have also made it much easier for end-users to design “buggy” surveys. Question order bias arises when placing one question before another leads to different responses from when their order is reversed[1]. Question wording bias occurs when different question variants inadvertently elicit wildly different responses[2]. When respondents abandon a survey partway through, this phenomenon is known as breakoff. Breakoff may be due to survey fatigue, when a survey is too long, or because a particular question is ambiguous, lacks an appropriate response, or is too intrusive. If an entire group of survey respondents abandon a survey, the result can be selection bias: the survey will exclude an entire group from the population being surveyed.

Inattentive or random respondents. Some respondents are inattentive and make arbitrary choices, rather than answering the question carefully. While this problem can arise in all survey scenarios, it is especially acute in an on-line setting where there is no direct supervision. To make matters worse, there is no “right answer” to check against for surveys, making controlling for quality difficult.

Downs et al. found that nearly 40% of survey respondents on Amazon’s Mechanical Turk answered randomly [6]. So-called attention check questions aimed at screening inattentive workers are ineffective because they are easily recognized. [1] While all of the above problems are known to practitioners [10], there is currently no way to address them automatically. The result is that current practice in deploying surveys is generally limited to an initial pilot study followed by a full deployment, with no way to control for the potentially devastating impact of survey errors and inattentive respondents.

The current approach to designing and deploying surveys is akin to writing a program, making sure it compiles, and then shipping it—to run on a system with hardware problems.

1.1 SurveyMan
SurveyMan is a lightweight tabular domain-specific language and runtime system for designing, debugging, and deploying surveys on the web, either by hosting it as a website, or via crowdsourcing platforms. The re-executable structure and dynamic analyses it performs provide a scientific footing for the development of surveys and the analysis of their results. The key idea behind SurveyMan is that by giving survey authors a way to write their surveys that steers them away from unnecessary ordering constraints, we can apply static analysis, randomization, and statistical dynamic analysis to locate survey errors and ensure the quality of responses.

Overview: Figure 1 depicts the SurveyMan workflow. Survey authors create surveys using the SurveyMan programming language. The SurveyMan static analyzer checks the SurveyMan program for validity and reports key statistics about the survey prior to deployment. If the program is correct, SurveyMan’s runtime system can then deploy the survey via the web or a crowdsourcing platform: each respondent sees a differently-randomized version. SurveyMan’s dynamic analysis operates on information from the static analysis and the results of the survey to identify survey errors and inattentive respondents. Table 1 summarizes the analyses that SurveyMan performs.

Domain-Specific Language. We designed the SurveyMan language in concert with social scientists to ensure its usability and accessibility to non-programmers [17]. Our approach leverages the tools that our target audiences use: because social scientists extensively use both Excel and R, which both feature native support for comma-separated value files, we adopt a tabular format that can be entered directly in a spreadsheet and saved as a .csv file.

SurveyMan’s domain-specific language is simple but captures most features needed by survey authors, including a variety of answer types and branching based on answers to particular questions. In addition, because SurveyMan’s error analysis depends on randomization, its language is designed to maximize SurveyMan’s freedom to randomize question order, question variants, and answers.

A user writing a survey with SurveyMan can optionally specify a partial order over questions by grouping questions into blocks, identified by number. All questions within the same block may be asked in any order, but must strictly
Static Analyses

<table>
<thead>
<tr>
<th>Well-formedness</th>
<th>Ensures survey is a DAG and other correctness checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>Ensures the possibility of seeing each question.</td>
</tr>
<tr>
<td>Survey statistics</td>
<td>Min, max, and avg. # questions in survey; finds short-circuits and guides pricing</td>
</tr>
<tr>
<td>Entropy</td>
<td>Measures information content of survey; higher is better</td>
</tr>
</tbody>
</table>

Dynamic Analyses

| Correlated Questions    | Reports redundant questions which can be eliminated to reduce survey length |
| Question Order Bias     | Reports questions whose results depend on the order in which they are asked |
| Question Wording Variant Bias | Reports questions whose results depend on the way they are worded |
| Breakoff                | Finds problematic questions that lead to survey abandonment |
| Inattentive or Random Respondents | Identifies unconscientious respondents so they can be excluded in analysis |

Table 1: The analyses that SURVEYMAN performs on surveys and their deployment.

Figure 1: Overview of the SURVEYMAN system.

SURVEYMAN system

SURVEYMAN static analysis (§3)

SURVEYMAN dynamic analysis (§5)

Web or Crowdsourcing Platform

Compile-time errors
Survey statistics

RESULTSB,CSV

Compile()me*errors*
Survey*sta}s)cs*

account the survey’s static control flow, the partial order on questions, and collected results. The survey author can then use the resulting report to refine their survey and exclude random respondents.

Note that certain problems with surveys are beyond the scope of any tool to address [19]. These include coverage error, when the respondents do not include the target population of interest; sampling error, when the make-up of the respondents is not representative of the target of interest; and non-response bias, when respondents to a survey are different from those who choose not to or are unable to participate in a survey.

Invariance The set of answers a respondent provides at a given point in time should be invariant under randomization. Since we cannot travel back in time, nor create parallel universes, we use statistical similarity to detect these differences. Wording bias and order bias are runtime errors that violate our response invariant.

2. Related Work

Standalone Survey DSLs. Blaise is a language for designing survey logic and layout [11]. Blaise programs consist of named blocks that contain question text, unique identifiers, and response type information. Control flow is specified in a rules block, where users list the questions by unique identifier in their desired display order. This rules block may also contain data validation rules, such that an Age field be greater than 18. Similarly, QPL is a language and deployment tool for web surveys sponsored by the U.S. Government Accountability Office [20]. Neither language supports randomization or identifies survey errors.

Embedded Survey DSLs. Topsl [8] and websperiment [9] embed survey structure specifications in general-purpose programming languages (PLT Scheme and Ruby, respectively). Topsl is a library of macros to express the content, control flow, and layout of surveys; Websperiment provides similar functionality. Both provide only logical structures for surveys and a means for customizing their presentation. Neither one can detect survey errors or do quality control.
XML-Based Survey Specifications. At least two attempts have been made to express survey structure using XML. The first is SuML; the documentation for this schema is no longer available and its specification is not described in its paper [1]. A recent XML implementation of survey design is SQBL [15] [16]. SQBL is available with a WYSIWYG editor for designing surveys [14]. It offers default answer options, looping structures, and reusable components. It is the only survey specification language we are aware of that is extensible, open source, and has an active community of users. Unlike SURVEYMAN, none of these provide for randomization or error analyses.

Web Survey Tools. Quite a few of these tools and services have recently added features similar to those provided by SURVEYMAN, such as question randomization. Some of these features are only available in premium versions of the software. None of the services available perform analyses on the quality of the responses, nor do they consider that the survey itself may be flawed.

Analysis of survey structure. Despite the fact that surveys are a well-studied topic—a key survey text from 1978 has over 11,000 citations [5]—there has been surprisingly little work on approaches to automatically address survey errors from a structural perspective. We are aware of no previous work on identifying any other errors beyond breakoff. Peytchev et al. observe that there is little scholarly work on identifying breakoff and attribute this to a lack of variation in question characteristics or randomization of question order, which SURVEYMAN provides [12].

Randomized Control Flow. As far as we are aware, SURVEYMAN’s language is the first to combine branches with randomized control flow. Dijkstra’s Guarded Command notation requires non-deterministic choice among any true cases in conditionals [4].

Crowdsourcing and Quality Control. Barowy et al. describe AUTOMAN, an embedded domain-specific language that integrates digital and human computation via crowdsourcing platforms [2]. Both AUTOMAN and SURVEYMAN have shared goals, including automatically ensuring quality control of respondents. However, AUTOMAN’s focus is on obtaining a single correct result for each human computation. SURVEYMAN instead collects distributions of responses, which requires an entirely different approach.

3. Contributions
The main contribution is the SURVEYMAN DSL. Because of its tabular format, SURVEYMAN users can enter their surveys directly in a spreadsheet application, such as Microsoft Excel. Unlike text editors or IDEs, spreadsheet applications are tools that our target audience knows well. SURVEYMAN is designed so that survey authors must go out of their way to state when randomization is not to be used. This approach encourages authors to avoid imposing unnecessary ordering constraints.

Surveys form a directed acyclic graph of groups of partially ordered questions. To create a basic survey, a survey author simply lists questions and possible answers in a sequence of rows; in this case, the graph would have one node, containing all of the questions. When the survey is deployed, all questions are presented in a random order, and the order of all answers is also randomized.

3.1 Syntax
Table 2 enumerates the recognized column headers; only QUESTION and OPTIONS are mandatory. The survey design may add columns, but these will not be used in the analysis. If a given column is not present, its default value is used. The columns may appear in any order. See [?] for the formal syntax.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUESTION</td>
<td>The text for a question</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Answer choices, one per row</td>
</tr>
<tr>
<td>BLOCK</td>
<td>Numbers used to partially order questions</td>
</tr>
<tr>
<td>EXCLUSIVE</td>
<td>Only one choice allowed (default)</td>
</tr>
<tr>
<td>ORDERED</td>
<td>Present options in order</td>
</tr>
<tr>
<td>BRANCH</td>
<td>For this response, go to this block</td>
</tr>
<tr>
<td>RANDOMIZE</td>
<td>Randomize option orders (default)</td>
</tr>
<tr>
<td>FREETEXT</td>
<td>Allow text entry, optionally constrained</td>
</tr>
<tr>
<td>CORRELATED</td>
<td>Used to indicate questions are correlated</td>
</tr>
</tbody>
</table>

Table 2: Columns in SURVEYMAN. All except the first two (QUESTION and OPTIONS) are optional.
EXCLUSIVE and FREETEXT columns are set. If there is only one option listed and the cell is empty, this question text is presented to the respondent as instructions.

Ordering. By default, options are unordered; this corresponds to nominal data like a respondent’s favorite food, where there is no ordering relationship. Ordered options include so-called Likert scales, where respondents rate their level of agreement with a statement; when the options comprise a ranking (e.g., from 1 to 5); and when ordering is necessary for navigation (e.g., for a long list of countries).

When options are unordered, they are presented to each respondent as one of the possible permutations. Ordered questions may appear as one of two possible permutations.

Branches. The BRANCH column provides control flow through the survey when the survey designer wants respondents who answer a particular question one way to take one path through the survey, while respondents who answer differently take a different path.

During the survey, branching is deferred until the end of a block, as Figure 2(a) depicts. By avoiding premature branching, this approach ensures that all users answer the same questions, regardless of randomization. It also avoids the biasing of question order that would result by forcing branches to appear in a fixed position.

Branching from a particular question response must go to a higher numbered block, preventing cycles.

3.2 Advanced Features

SURVEYMAN has several advanced features to give survey authors more control over their surveys and their interaction with SURVEYMAN.

Correlated questions. SURVEYMAN lets authors include a CORRELATED column to assist in quality control. SURVEYMAN checks for correlations between questions to see if any redundancy can be removed, since shorter surveys help reduce survey fatigue. However, sometimes correlations are desired, whether to confirm a hypothesis or as a form of quality control. SURVEYMAN thus lets authors mark sets of questions as correlated by filling in the column with arbitrary text: all questions with the same text are assumed to be correlated. SURVEYMAN will then only report if the answers to these questions are not correlated. If not, this information can be used to help identify inattentive or adversarial respondents.

Advanced Blocks Notation

Blocks have types that are represented by 2-tuples of their branching policies and their location flags. The branching policies are NONE, ONE, and ALL. NONE denotes a block that exhibits no branching behavior. ONE denotes a block that has exactly one branching question. ALL denotes a block whose contents are question variants; its branching behavior is effectively random sampling. We provide a dummy target (NULL) for ALL blocks that don’t have a true branching target. This defers selection of the branch target until runtime.

The location flag refers to the block’s ability to be randomized: floating blocks may move around inside their parent blocks, while stationary blocks may not. We describe each in terms of their interaction with other blocks and with branches. Figure 2(b) gives an example of each.

If the survey is completely flat and there are no blocks, we say that all questions belong to a single, top-level block (block 1). To ensure our survey invariant, we enforce the restrictions that (1) branch question targets be top-level stationary blocks (or NULL) and (2) floating blocks may have type ONE, nor ALL with a non-NUL target.

3.3 Static Analyses

After successfully parsing its .csv input, the SURVEYMAN analyzer verifies that the input program itself is well-formed and issues warnings if it finds any unrecognized headers, and if it finds any duplicate questions. It then performs more detailed analysis of branches and blocking to ensure that the survey invariant is maintained. It also checks for cycles in the survey, and that all blocks are reachable.

Survey Statistics and Entropy If a program passes all of the above checks, SURVEYMAN produces a report of various statistics about the survey, including an analysis of the minimum, maximum, and average path lengths through the survey. This data is used to recommend pricing.

SURVEYMAN also reports the entropy of the survey. This number roughly corresponds to the complexity of the survey. If the survey has very low entropy (few questions or answers), the survey will require many respondents for SURVEYMAN to be able to identify inattentive respondents. Surveys with higher entropy provide SURVEYMAN with greater discriminatory power. For a survey of \( n \) questions each with \( m_i \) responses, SURVEYMAN computes a conservative upper bound on the entropy on the survey as \( n \log_2(\max\{m_i \mid 1 \leq i \leq n\}) \).

3.4 Compiler and Runtime System

The SURVEYMAN compiler transforms its input into a payload that runs inside a web page. SURVEYMAN then deploys the survey, either by acting as a webserver itself, or by posting the page to a crowdsourcing platform and collecting responses. Display of questions and flow through the survey are managed by an interpreter written in JavaScript. As responses arrive, SURVEYMAN performs its dynamic analyses described.

SURVEYMAN transforms the survey into a minimal JSON representation wrapped in HTML, which is executed by the interpreter. Each user sees a different ordering of questions. However, each particular user’s questions are always presented in the same order. SURVEYMAN displays only one question at a time. This design decision is not purely aesthetic; it also makes measuring breakoff more precise, because there
We evaluated S
After collecting all results, S
(a) In this example of survey branches, the respondent is asked the branching question (Q3) as the second question. The execution of this branch is deferred until the respondent answers Q2, the last question in the block.

(b) Two different instances of a survey with top-level blocks (1 and 2), nested blocks (1.1 and 1.2), and a floating block (1.B). The floating block can move anywhere within block 1, while the other blocks retain their relative orders (1.B).

Figure 2: Examples of branches and blocks.

is never any confusion as to which question might have led someone to abandon a survey.

3.5 Dynamic Analyses
After collecting all results, SURVEYMAN performs a series of tests to identify survey errors and inattentive or random respondents. These are standard tests used by social scientists, adapted to SURVEYMAN’s unique properties.

3.6 Case Studies
We evaluated SURVEYMAN’s usefulness in a series of case studies with surveys produced by our social scientist colleagues. See the SURVEYMAN paper [17] for details of the case studies.

References