Boot Camp or Bordello: Whipping Rookies into Shape

June Bradley
Bradley Mann Consulting

Gerard McGrath
Macquarie University

Follow this and additional works at: http://aisel.aisnet.org/icis2000

Recommended Citation
http://aisel.aisnet.org/icis2000/46

This material is brought to you by the International Conference on Information Systems (ICIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in ICIS 2000 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
BOOT CAMP OR BORDELLO: WHIPPING ROOKIES INTO SHAPE

June Bradley
Bradley Mann Consulting
Australia

G. Michael McGrath
Macquarie University
Australia

Abstract

The overhead associated with training new staff brought onto software development teams has long been a matter of concern for both information systems (IS) researchers and practitioners. The projected global shortfall in qualified IT professionals in coming years only serves to emphasize the importance of this issue. This is particularly so, given that it is generally recognized that training overhead may outweigh any benefits derived from adding new members to a project team.

Thus, the issue of on-the-job training has received plenty of attention from IS researchers. There is, however, a notable lack of useful prescriptive guidelines in the literature and practitioners tend to rely very much on heuristics (developed largely from individual experience). Our own view is that most commonly-used estimates for veteran training efforts and rookie assimilation are too low. In addition, we believe that a much finer-grained level of detail is required in any model proposed as a useful decision support aid in this area. At the very least, we contend that such a model should encompass project and organizational characteristics; rookie experience, ability, and skills match with project requirements; and, following from this, training needs.

To test this, we have embarked on a series of case studies. Stage 1 is exploratory and developmental, major aims being the development of a system dynamics model of on-the-job programmer training and the generation of a set of hypotheses. Stage 2 involves the testing of these hypotheses. Essentially, the systems dynamics model is a representation of Stage 1 findings. Typically, in systems dynamics models, most variance is caused by a limited number of parameters. Thus, by identifying these and employing replication logic, we can construct a set of Stage 2 field studies that we can use to validate, refute, or refine our Stage 1 findings.

Keywords: IS training, IS development, IS turnover

1. INTRODUCTION: MOTIVATION AND THEORETICAL FOUNDATIONS

Staff turnover and training of replacements have long been issues of concern for both IS researchers and practitioners. Willoughby (1977) reported turnover rates of 15% to 20% (per annum) during the 1960s, Tanniru and Taylor (1981) estimated that this figure had increased to 25% by the late 1970s and, according to Bott (1982), the rate had increased to 34% by the early 1980s.

While most researchers have recognized the cyclical nature of staff turnover, some more recent reports suggest that things could deteriorate further before any improvement is evident. Specifically:

• A recent study (Computerworld 1999) found that nearly 300,000 (10%) of U.S. IT service and support jobs were vacant, 77% of IT companies were having difficulty finding qualified staff and that the current IT staff shortage is costing the U.S. economy $4.5 billion annually;
Bradley and McGrath

• On average, U.S. programmers only stay in a job for three to four years (Earls 1998); and

• In Australia, the estimated growth in demand for IT specialists is 9% over the next 12 months, 25% over the next three years and 50% over the next five years (Deloitte 1999).

Thus, staff replacement is a matter of vital concern for project managers. More than 20 years have now passed since Brooks (1978) identified time lost in veterans training rookies as a major factor underpinning his famous law: “Adding manpower to a late software project makes it later.” However, Abdel-Hamid and Madnick (1991) have noted that, insofar as veterans’ time taken up in training rookies is concerned, they were unable to find any proposed formulae in the literature and they also found that practitioners tended to rely on rules of thumb—ranging from 15% to 25% (for the complete assimilation period). Furthermore, estimates for the assimilation period itself vary greatly. Abdel-Hamid and Madnick, again, quote a range of estimates between two and six months and, in their own model, split the difference and set their “assimilation delay” variable to 16 work-weeks.

Our view is that these commonly-used estimates (for veteran training efforts and rookie assimilation) are too low. In addition, we believe that a much finer-grained level of detail is required in any model proposed as a useful decision support aid in this area. At the very least, we contend that such a model should encompass project characteristics; rookie experience, ability, background, and skills match with project requirements; and, following from this, training needs. These beliefs underpin our research and, somewhat rephrased, are the basis for our research questions and hypotheses.

2. RESEARCH DESIGN

Our principal research questions are:

What training overhead (and how much) should be allowed for when bringing new programmers into a project team?

and

What are the major factors responsible for this training overhead, how is it related, and what are the implications of these factors and relationships?

Much staff training research has focused on overhead (Bott 1982; Corbato and Clingen 1979; Curtis et al. 1988), when (and how much effort) to commit to training new starters (Brooks 1978), delays in adjusting team size (McLaughlin 1979) and assimilation (Brandon 1970). However, most work has tended to concentrate on one or two factors in isolation and has largely ignored the complex web of dependencies, relationships, and feedback loops that can have major impacts on the total system. Moreover, recent work appears to have added little in the way of major refinements to the work of the early researchers in the area. An exception is the system dynamics-based approach of Abdel-Hamid and Madnick (1991) and our research is based very much in this tradition.

Our research is based on a series of case studies, conducted in two stages: Stage 1 is exploratory and developmental, the major aims being the development of a system dynamics model of on-the-job programmer training and the generation of hypotheses. Stage 2 involves the testing of these hypotheses. Essentially, the systems dynamics model is a representation of the key findings of Stage 1. Typically, in systems dynamics models, most variance is caused by a limited number of parameters. Thus, by identifying these and employing Yin’s (1994) replication logic, we can construct a set of Stage 2 field studies that we can use to validate, refute, or refine our Stage 1 findings.

Currently, we are in Stage 1 of our research. Our principal source of data are archival records. Here, we were fortunate to come into possession of a set of time sheet records maintained by a freelance programmer over a 10-year period. These records detailed her experiences on a total of 18 projects, at eight sites, ranging from very large to small, developed mostly in COBOL, and undertaken on a range of platforms. She kept and maintained detailed records of time she spent training recruits during these projects. Moreover, she developed the training classification scheme detailed in footnote 1 and assigned her time within this

1We divide training into the following four categories: (1) Administration, encompassing staff introductions and briefings, essential details on organization policies and procedures, organizational norms and cultural values, Personnel Department needs and local policies and procedures; (2) Operating Systems, concerned with learning the operating system(s); (3) Environment, including site-specific procedures related to coding, testing, documentation, configuration management, quality assurance, etc.; and (4) Applications, concerned with essential details (e.g., functionality and data models) of the application under development.
framework. We are employing this rich source of data to build our initial model and, at this point, we have organized, interpreted, and analyzed data from our first two projects (as well as having developed a preliminary version of our model). Data from all remaining projects will be used to refine, extend, and test the initial model.

The system dynamics model serves several purposes: (1) it is, effectively, the Stage 1 case study database; (2) it represents key dependencies and relationships between study variables; (3) the powerful simulation capabilities of the software package used enables extensive analysis of the Stage 1 data; and, finally, (4) simulation results are intended as the key source of data for derivation of the hypotheses to be tested during Stage 2.

The use of a single source of data for Stage 1 poses an obvious threat to the external validity of our results: namely, other programmers might be more or less effective as trainers than our source and this could influence both our model’s behavior and our preliminary conclusions. However, the technology that underpins the model permits us to determine just how sensitive it is to such variations in programmer capabilities and, more importantly, we recognize the need to test our preliminary findings through further case studies.

3. A SYSTEM DYNAMICS VIEW

Figure 1 displays a model of our problem domain developed using the system dynamics modeling tool, ithink (HPS 1994). Our model focuses on “on-the-job” training during project development. Generally, this training is one-on-one, where an experienced team member (a veteran) imparts essential project knowledge to a newcomer (a rookie). It is assumed that very little in the way of (immediately) useful work results from this training.

The basic building blocks of system dynamics models are stocks (represented as rectangles), flows (represented as arrows with circular flow regulators attached), and converters (represented as circles). In our model, examples of stocks are Prod Vet Hrs (productive veteran hours) and Rookie Prody Index (rookie productivity index). There is a level associated with each stock, which can be an actual value (as in the case of Prod Vet Hrs) or a value bounded by some artificial scale. Rookie Prody Index, for example, is represented on a zero-to-one scale and measures a rookie’s effectiveness in relation to a veteran. For example, a value of 0.75 would indicate that a rookie (being trained) is only 75% as effective as the veteran (doing the training). Stock levels vary with flows, which may be inflows, outflows, or bidirectional. For example, WklyVProdHrs (weekly veteran productive hours) is a uni-directional inflow, such that:

\[ \text{Prod Vet Hrs}_t = f(\text{Prod Vet Hrs}_{t-1}, \text{Wkly Tngt}, \text{Wkly_Hrs}). \]

Specifically, the productive hours for a veteran at time \( t \) are determined by adding the difference between the weekly hours available (37.5) and the total time spent in rookie training to the productive veteran hours level at time \( t-1 \). These equations are the foundation of ithink’s formidable simulation capabilities. The third of our basic constructs, converters, serves a utilitarian role: they hold values for constants, calculate mathematical relationships, and serve as repositories for graphical functions.

At this stage, our model is not particularly complex. Base data for one case study (at a time) is used to populate the RVTng and Individual Tng converters in our model. We may now calculate individual and cumulative values for Prod Vet Hrs (essentially, the time the veteran is not involved in rookie training). A rookie’s productive hours (Prod Rookie Hrs) is calculated in exactly the same way, except that time spent in individual training (or waiting for a veteran’s help) must also be subtracted from the total available time (Wkly Hrs). From here, calculation of the percentage of time that both veterans and rookies have available for productive work (RProdRatio and VProdRatio) is a trivial exercise.

The assumption is made that a veteran produces 20 lines of fully-tested code (LOC) per work-day. OS Code (outstanding code) is initially set to a level sufficient to keep two uninterrupted veterans fully-occupied for the duration of the project under investigation. During each simulated work-week, the OS Code stock is depleted and the VetOnly Code and Vet&Rookie Code stocks increase at rates determined by their coding productivities: respectively, (1) 20 LOC/work-day (VetOnly Prod) and (2) VetOnlyPrody adjusted to take into account both lost time (determined by the veteran and rookie productivity ratios) and limited rookie effectiveness while being trained\(^2\) (as determined by the Rookie Prody Index). By establishing our model in this way, we may directly compare the effectiveness of a veteran working alone and a veteran and rookie working together.

---

\(^2\)Along with Abdel-Hamid and Madnick (1991), we set initial rookie effectiveness at 0.5 and assume that it takes 16 work-weeks for a rookie to come up to speed. This does not mean that at the end of 16 weeks a rookie is as useful as a veteran, only that a rookie is as productive as a veteran in the tasks for which he or she has been trained.
4. CASE STUDIES

We have labeled our first two cases the “boot camp” and the “bordello.” The boot camp was so-named because new starters received about as much care, consideration, and attention as the average army recruit. On the other hand, the bordello catered for clients who were paying big money and it was made very clear to our programmers that no amount of client pampering and stroking was too much—and no fee too high!

Both projects involved insurance applications. Bruce joined the boot camp about two-thirds of the way through an 18 month project, but only had limited experience with insurance applications. The development environment was very informal and, in CMM (Capability Maturity Model) terms, would probably not rate at much more than Level 1. Louise was assigned to the bordello to complete 17 weeks of predetermined, but routine, maintenance work. She had extensive experience with insurance
applications, but had limited exposure to the IBM mainframes employed. Development processes in the bordello were considerably more formal than those at the boot camp (around CMM Level 2).

Simulation output produced by our model demonstrates that, during the 17 weeks Louise spent in the bordello, at no point were the rookie and veteran combined producing more code than would have been generated by the veteran alone. Indeed, at the end of week 17, the combined veteran-rookie productivity was only 92% of the estimated output of a single veteran. Bruce’s experience at the boot camp was similar. While at the end of week 22 he and the veteran were 1.23 times as productive as a single veteran, the veteran alone would have produced 23% more code over the full 22 weeks of Bruce’s contract.

Breakdowns of training time into our four categories at the boot camp and bordello are presented in Figures 2 and 3 respectively. Figure 2 clearly highlights Bruce’s lack of experience with insurance applications (because of the dominance of category 4—still rising at the end of 22 weeks) and Figure 3 equally demonstrates Louise’s limited knowledge of the IBM mainframe development environment (category 2 dominance). These results provide fairly convincing support for our contention that a relatively fine-grained level of analysis is required when assessing the impacts of rookie training. Indeed, had these simulations been run in advance, the substantial impacts of what probably appeared to be minor mismatches between project requirements and our rookies’ skill-sets may well have become readily apparent.

5. CONCLUSION

Early indications are that our research may produce yet another demonstration of the negative impact of staff turnover and training on system development productivity. While this has long been recognized, little in the way of prescriptive guidelines is available to assist the unfortunate project manager faced with tight deadlines, key staff losses, and untrained replacements. Moreover, simple heuristics are inadequate: for a prescription to be truly useful, it must encompass vital contextual information associated with a project and its development team. Our major study objective is to produce a model of this turnover/training domain that is useful as both an explanatory and prescriptive aid. Using system dynamics, we intend to structure and model a rich source of field data and simulate productivity outcomes for a variety of circumstances. Because the archival data used in Stage 1 of our research comes from a single source, the external validity of any conclusions must be suspect. However, we expect that this exploratory work will produce well-grounded hypotheses that we may then test in later field studies—probably via action research.
References


The following reference list contains URLs for World Wide Web pages. These links existed as of the date of submission but are not guaranteed to be working thereafter. The contents of Web pages may change over time. Where version information is provided in the References, different versions may not contain the information or the conclusions referenced. The author(s) of the Web pages, not ICIS, is (are) responsible for the accuracy of their content. The author(s) of this article, not ICIS, is (are) responsible for the accuracy of the URL and version information.