

Appendix 3-1- Reflections on the data collection and modeling process

In this document I briefly discuss the process used for data gathering, model building, and model analysis that I employed in the research for the paper “Dynamics of platform-based product development”. I focus mainly on a few tools and activities that I found useful for myself and I had not found discussed in detail in the literature. Therefore I leave the comprehensive description of the qualitative research process and model building to relevant books and articles (Glaser and Strauss 1967; Eisenhardt 1989; Strauss and Corbin 1998; Sterman 2000) and keep this document short and operational. I make no claim of the efficiency of the processes I have used, rather I will simply describe my experience and highlight process insights of this experience. The intended audience of this appendix is other researchers who want to build theory based on case studies using simulation, as well as model builders who look for doing their testing and analysis faster and more efficiently.

This research started in June 2004 and has continued in multiple steps, outlined chronologically below:

- June-August 2004: Interviews, archival data gathering, and hypothesis generation
- July 2004: building a qualitative model
- August 2004: Building a detailed simulation model and preliminary analysis
- September-October 2004: Model analysis, building a simple insight model
- November 2004: First write up
- Dec 2004-June 2005: Further data collection (both interviews and quantitative data), and refinement of analysis and text

- March 2005 on: Definition of other research questions in the research site and data-gathering for these questions.

Four main activities are evident in this process: data gathering, model building, model analysis, and model distillation/insight generation. The process of moving between these activities was iterative, with the first iteration being the most intense.

Data gathering and qualitative model building

In the data gathering phase, the main source of information was interviews with individuals involved with the development of two software products, Alpha and Beta. I entered the site with some general impression of the problem I wanted to study: the quality, cost, and delay problems faced by many new product development projects, when multiple projects are connected to each other. These were very real challenges for product Alpha, and product Beta was distinguishable for avoiding several of these problems¹. Therefore the contrast between the two seemed promising for making sense of relevant issues.

Interviews were open ended and explored different themes relevant to the problem at hand, as well as the mechanics of the development process and the relationships between different groups involved. Interviews were often recorded and summarized at the end of the day in daily report files. In essence, these summaries were shortened transcripts of the interviews which I jotted down while listening to the audio files at the end of each day. Marking each topic of conversation with a header and a time stamp (which minute and second of the interview it came about) helped accessibility of these files for future

¹ In fact I started with product Alpha and added product Beta to my study when in mid July (04) I learnt about its significantly better practices.

reference, while keeping their size small and therefore their review more feasible. An example of these transcripts is reproduced below:

“13:50 Translating features

Product manager comes up with features, gives to system engineers, which for them have been fine in Siebel, since the same person has been doing system engineering across multiple releases and was a well-understood area. So we got most of requirements well. We still find problems on customer expectation not being the same, but that is usually the problem of VPs thinking that we need a desktop application ([person A] and [Person B]) and pushing this, while this ended up appear not to be central at all

17:00 Feature additions and drops

We flip flop a little bit. QQQ it was driven mostly by thinking that we can sell it with higher price, but customers didn't want it. We have so much problem getting more than one application with Siebel on the screen that is problematic.”

As the example shows, each paragraph tracks 2-3 minutes of conversation in a couple of raw sentences, moreover, I did a quick coding for action items, data sources mentioned, good quotations, etc while doing the transcription, by leaving marks such as QQQ in the text, where relevant. These were easy to automatically code in the Atlas.ti software that I used for qualitative analysis and allowed me to quickly extract important action items, people to interview, etc, at the end of the week. This transcription and coding process took about 1.5 times of the original interview. In retrospect coding for action items and

data sources are helpful, but too much coding of content at this early stage is not so helpful.

Along with the data-gathering process, I built a qualitative model (in causal loops) of dynamics that can impact multiple release product development. The data-gathering and model building activities were intimately inter-related: interview data provided new hypothesis to be added to the loop-set, while, the loop-set guided the questions to be asked for the confirmation of existing dynamic hypotheses and for the exploration of the new relevant themes. Moreover, often the loops played a valuable role as boundary objects to discuss the research with interviewees and to quickly elicit their comments on other dynamics relevant to the problem at hand. Appendix 3-2 outlines the main dynamic hypothesis generated from this part of process through causal loop diagrams.

Simulation model building

The qualitative model discussed above informed the formulation of a simulation model for multiple release product development. I started with the core mechanics of product development for a single release, added some of the more salient feedback processes from the causal loop diagrams, and then expanded the model to include multiple releases and the interactions between these releases. Future refinements included more feedback loops and hypotheses to this model. The detailed model formulations are listed in the appendix 3-3.

The simulation model resulting from this process was detailed (over 1500 variables and parameters) and complex. Therefore the model required a strict discipline to ensure the integrity and the consistency of the model after frequent additions of new pieces of

structure. This need was further pronounced by the tight time-line of model building on which I was working (about 1 month to build the detailed model from scratch). A set of automated tests facilitated such robustness analysis, which I describe below. This testing process is in nature quite similar to the “Reality checks” feature of Vensim, however, I found the implementation of the latter relatively confusing and therefore followed the following procedure.

In short, I created a set of test cases, built them into a command script that executed the relevant simulations consecutively, and ran the command script upon each modification of the model. I then inspected a few relevant metrics in the control panel view of the model to see if the model has broken down as a result of the last modification.

Design of the test cases- The correct behavior of the model under a set of extreme conditions and theoretical scenarios is often predictable. For example, in absence of development resources, no task will get done. These cases create the basis for automated testing. Below is a few of such cases that I used to test my model:

- *When there is no work to do, nothing should happen*
- *When there is no resources to work, nothing should happen*
- *When we have unlimited resources for testing and development, the only bottleneck is design and testing time, therefore each stage should take as much as design time plus a small amount for first order finishing of tasks in development and testing stages and the circling of tasks between rework and testing.*
- *When we install in many sites, we find most of the errors very fast. If then the resources for CE are unlimited and all errors are considered important, then all the errors are quickly fixed and no errors remain.*

- *When we start all releases at the same time (0), nothing strange should happen with the scheduled times to finish etc.*
- *When we put all the problematic feedback loops out of picture, things should go smoothly*

Each test case needs a condition that can be translated into parameter changes, and an expected behavior which is easily inspected in the simulation output.

Building the command script- DSS version of the Vensim software has the capacities for writing command scripts, pieces of code for controlling and using the software based on a pre-defined scheme. Therefore I wrote the command scripts that set up the model based on each test-case scenario, and simulate it with a specific run name. See an example below (the first line includes the test description).

! When we put all the problematic feedback loops out of picture, things should go smoothly

SIMULATE>SETVAL/SW Eff Archctr Complexity on Productivity = 0

SIMULATE>SETVAL/SW Eff Old Arch on Design Problems = 0

SIMULATE>SETVAL/SW Eff Old Code and Archctr on Error = 0

SIMULATE>SETVAL/SW Eff Pressure on Error = 0

SIMULATE>SETVAL/SW Eff Pressure on Productivity = 0

SIMULATE>SETVAL/SW Eff Bug Fix on Scope=0

SIMULATE>SETVAL/SW Effect Replan on Replan Treshold=0

SIMULATE>SETVAL/SW Fixing Bugs=0

SIMULATE>SETVAL/SW Endogneous Sales=0

SIMULATE>RUNNAME/TL1-NoFeedbackActive.vdf|O

MENU>RUN/O

Running the command scripts and inspecting the model behavior- Testing often leads to finding problems in the model that need to be fixed. To avoid problems of version control (having multiple versions of the model which each have fixed some aspects of the problem), I copied the latest version of the model into a testing directory (which includes all the relevant command files), ran the tests on that version of the model, and edited the model according to the problems found. When finished with this round of testing, I returned this edited version of the model to the main directory and saved it under a new version name. I could then build on this model for future work.

I used a control panel which included several of the critical variables of the model to track the behavior of the model in automated testing. Moreover, a few variables (such as the “minimum of all physical stocks”) were added to enable the faster inspection of the test results. Meaningful run names help quickly inspect each test run for potential flaws and go to the next in absence of any flaws.

The concept of extreme condition testing is well-established in SD literature (See Chapter 21 in Sterman 2000). However, time constraints often cut down on the extensiveness of their application. Automated testing is very quick in running a set of pre-defined tests on the model and therefore allows the modeler to do so very regularly and ensure the robustness of model to extreme conditions and different scenarios, with a little up-front time investment. In fact, for a relatively complicated model, the establishment of the test cases and command script took me 2-3 hours, and afterwards each round of testing was very fast (few minutes).

Another useful practice in the modeling phase includes the color coding of different variables in the model. Color coding of variables helps the modeler navigate different views much faster, run different tests easier, and make sense of different structures quicker. The basic idea is to have distinct colors for different types of variables (e.g. parameter, data source, switch, table function, etc). The following color codes were developed by MIT Ph.D. students in summer 2004² and account for visibility of parameters when in sensitivity analysis modes of Vensim software. Other modelers are encouraged to use a similar scheme to improve compatibility and avoid cognitive switching costs. Also short prefixes mentioned below can help organize the variables better (e.g. SW for switch).

Green Exogenous Parameters, Data: Data series, Parameters for table functions

Blue TL: Table functions

Pink SW: Switch

Purple TST: Test parameters/structural parameters/exogenous

Orange Output variables, OPT: Payoff variables

Brown IN: Initial conditions

Red AF: Attention flags

Model analysis and simplification

Much is written about different facets of model analysis. Here I focus on a quick derivation of behavioral loop dominance analysis (Ford 1999) which can be automated within available capabilities of Vensim software. While this method is not as rigorous as eigenvalue analysis and pathway participation methods (Forrester 1982; Eberlein 1989;

² Thanks to Jeroen Struben and Gokhan Dogan for their contributions to this development.

Mojtahedzadeh, Andersen et al. 2004), it has the benefit of being already operationally available for large models, with some fruitful results. I used this method to create a quick ranking of more important feedback loops from a larger set of hypothesis in the detailed simulation model.

The basic idea is to measure the sensitivity of a few basic metrics to existence of different feedback loops in the model. Ranking of the loops in their impact on these metrics determines which ones we want to focus on for further analysis. To do so we need to:

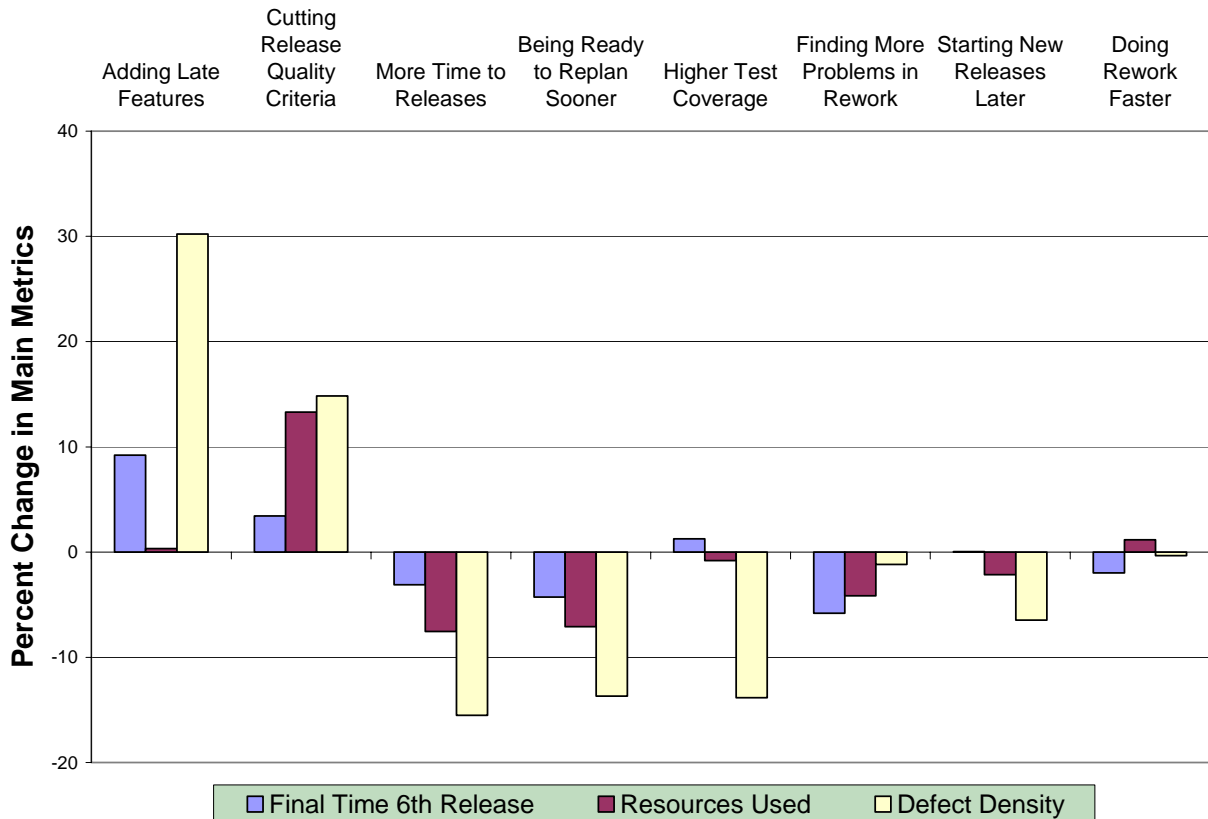
- Define a set of metrics about which we care (for example the cost and budget over-run) and implement these in the model as explicit variables.
- Run a sensitivity analysis switching off all the important loops in the hypothesis set, one at a time.
- Observe the sensitivity of different metrics to these loops and rank-order the loops based on these sensitivities to find the more important hypotheses.

Vensim allows us to perform the above-mentioned sensitivity analysis very fast (in seconds). For this purpose you need to have switch parameters defined for each loop in the hypothesis set, which cut the loop off the simulation (See Ford (1999)). It is helpful to name and parameterize these switch variables consistently, i.e. all of them should start with similar prefix and loops should be cut when their switch is 1 (or 0). With these settings in place, you can use the optimization option of the Vensim, include your metrics in the payoff function, turn the optimizer off, put sensitivity to “All Constants=1”, and simulate the model. Such simulation will automatically change all the parameters of the model, one at a time, 1% upper and lower than their base value, and reports the changes in the payoff from the base case. The results will be recorded in two .tab files, sensitiv.tab

and sortsens.tab, one sorted alphabetically and the other sorted based on the extent of sensitivity of payoff to the parameters of the model.

Since the switches in your model are put at 0, they will be changed to 1 and -1, the former of which is equivalent to cutting the feedback loop. Therefore the analysis automatically cuts every feedback loop (that has a switch variable) and observes the impact on the metric of interest. You can then extract the subset of results which are related to loops by copying those records from the alphabetically sorted “sensitive.tab” (remember that same prefixes were used for all the switch variables) and analyze them in a spreadsheet. This process can be further automated within a command script that changes the payoff function to investigate the sensitivity of all different relevant metrics and extract the results into a spreadsheet.

The main goal of this method is to simplify the process of selecting a set of more important hypotheses to include them in smaller models, in write ups of the study, or in the communication with the client team. This was a salient problem in my study, where I wanted to distill the most critical dynamic hypotheses from a list of over a dozen loops captured in the detailed model discussed above. The strength of the impact of different loops on the behavior of interest is an important consideration for such choices. This method facilitates inclusion of this consideration more robustly. The graph below shows an example of the final results that can come from this analysis.



Note that this procedure is not without problems. First, the set of feedback loops you choose are often not comprehensive. There are algorithms to select an independent and comprehensive loop-set (Oliva 2004) that can relieve this shortcoming. Nevertheless, the main goal of the above method is not to find the dominant loop in the structure, but to find the relative importance of a few different hypotheses (loops). Second, this analysis does not investigate the interactions between loops because it cuts loops one at a time. It is possible to include interactions by doing a full-grid sensitivity analysis of switch parameters with their values set at 0 or 1. Such analysis, however, increases the computational burden to 2^N (from N) and needs further data reduction methods (e.g. regression) to be applied on the outcome data to allow a ranking of loops from such multi-dimensional dataset. Finally, this method requires the definition of relevant metrics,

which can be a complicated task in some settings (for example when we care about oscillatory behavior). In short, this method is a quick and dirty way of getting an impression on relative importance of main hypotheses in their impact on a few metrics of interest. It is neither comprehensive, nor flawless, however, in the real world applications where limited time often impacts the quality of analysis, it can prove useful.

Another helpful practice in analysis phase includes examination of the results of a full sensitivity analysis on all model parameters (as described above) one by one, guessing the impacts on each of the metrics and then observing the real impact, and trying to explain the discrepancies between our intuition and the observations. Doing this for the top (most important) 20% of parameters often results in a very good understanding of the model behavior and its origins.

Model distillation

Case study and qualitative hypothesis generation often push the modeler to build relatively complex models which include hundreds, if not thousands of variables, and are hard to maintain, understand, and communicate. Above, I discussed a few tools I used to facilitate the maintenance, testing, and understanding of one such model. A critical next step of this process is to distill the critical dynamics of the model into a simple model which can be used to communicate the results, right papers, and do more comprehensive analysis or analytical work on the model. The model documented in the paper (“Dynamics of platform-based product development”) is in fact a distillation of the detailed model discussed above, which allowed for reducing the number of variables over 20 times, while keeping most of the critical dynamics.

Contrary to intuition, in my experience, it was probably harder to formulate the latter type of the model (small insight model) directly from the detailed data provided through a case study. Taking the intermediate step of constructing a larger and more detailed model allowed me to bridge the gap between the nuanced qualitative data, and the sweeping aggregations and abstractions required for the insight model. Besides, the detailed model can play other roles in calibration, management flight simulator development, and getting buy-in from the client. The practices discussed above allowed me to speed up the process and make it feasible to do both types of models in a relatively short time span (about 4-5 months of full time work to go through one round of this process), therefore I hope they can prove helpful for some other researchers.

References:

- Eberlein, R. L. 1989. Simplification and understanding of models. *System Dynamics Review* **5**(1).
- Eisenhardt, K. M. 1989. Building theories from case study research. *Academy of Management Review* **14**(4): 532-550.
- Ford, D. N. 1999. A behavioral approach to feedback loop dominance analysis. *System Dynamics Review* **15**(1): 3-36.
- Forrester, N. 1982. A dynamic synthesis of basic macroeconomic theory: Implications for stabilization policy analysis. *Sloan School of Management*. Cambridge, MA 02142, MIT.
- Glaser, B. G. and A. L. Strauss. 1967. *The discovery of grounded theory; strategies for qualitative research*. Chicago,, Aldine Pub. Co.

Mojtahedzadeh, M., D. Andersen and G. P. Richardson. 2004. Using digest to implement the pathway participation method for detecting influential system structure.

System Dynamics Review **20**(1): 1-20.

Oliva, R. 2004. Model structure analysis through graph theory: Partition heuristics and feedback structure decomposition. *System Dynamics Review* **20**(4): 313-336.

Sterman, J. 2000. *Business dynamics: Systems thinking and modeling for a complex world*. Irwin, McGraw-Hill.

Strauss, A. L. and J. M. Corbin. 1998. *Basics of qualitative research : Techniques and procedures for developing grounded theory*. Thousand Oaks, Sage Publications.