Massachusetts Institute of Technology
Sloan School
Applications of System Dynamics
15.875

Prof. Jim Hines
Spring 2004

Guidelines for sixth project week and presentation: Model and analysis of first hypothesis[1]

Note: This is the last of the handouts decribing the standard method. After this handout, choose another dynamic hypothesis and go to work – that is, from this point on you will continue cycling through handouts 5 and 6.

The sixth week: In the sixth week you should complete the computer simulation model of your first hypotheses and *analyze* its behavior. Analysis involves identifying model structure responsible for interesting model behavior, deciding whether that structure also exists in the real world, and figuring out what to do about that structure in the real world. For example, perhaps the structure in question causes oscillations in the model; you decide a similar structure (i.e. a similar loop) exists in the real world. In this case, you should develop some suggestions for what could be done to reduce the amplitude or likelihood of oscillations in the real world.

Caution: Be sure to allow sufficient time for analysis. As a rule of thumb figure that analyzing the model will take about as much time as building the model.

In the presentation, please provide an in-depth look at this first hypothesis – explaining its dynamics, why the dynamics are relevant (or why you originally thought the loop(s) would produce relevant dynamics), and the implications that the loop and its dynamics have for your client.

Analysis. First simulate your model. If you are "lucky", the behavior will surprise you. Find the structure (often a single loop) responsible for the surprising behavior. Causal tracing will identify the structure responsible for most non-oscillatory modes. Oscillatory modes will yield to loop-knockout and loop-isolation techniques. People who have taken 15.876 may want to use an eigenvalue approach in addition.

Next go through your model parameter by parameter. For each parameter

1. Guess how the model will behave if you double the parameter.
2. Double the parameter and simulate.

[1] Prepared 1998 by Jim Hines. Revised March, 1999, July, 1999,March 2000, April 2004.

3. Understand any surprise using causal tracing, loop knockout, or loop isolation.
4. Guess how the model will behave if you cut the parameter in half.
5. Cut the parameter in half.
6. Understand any surprise using causal tracing, look knockout, or loop isolation.
7. Set the parameter back to its original value.
8. Go to step 1 for the next parameter.

Causal tracing. Causal tracing is usually effective at identifying the structure responsible for non-oscillatory modes. To perform causal tracing, begin by looking at the simulated behavior of the immediate inputs to the variable that is behaving surprisingly. Identify which of the input variables is responsible for the surprising behavior. Now, continue the causal trace by looking at inputs to this "new" variable. Identify which input is responsible for the surprising behavior of the "new" variable. Keep going until you understand the structure causing the behavior (often this will be when you complete tracing a loop). This process is easy using the strip graph tool. (Note you will need to keep track of the variables you are tracing in order to know when you have completed a loop).

Loop Knockout. Causal loop tracing is less effective for oscillatory modes. A brute force method is often required to find the structure responsible for an oscillatory mode. Identify a likely negative loop. Disable it (i.e. knock it out). Be careful that you only knockout the loop you intend. Does the model still oscillate? If not, that loop is likely the culprit. To make sure, disconnect that loop from all other loops (you might need to create a new model) and simulate – does that loop oscillate on its own? If so, you have found the oscillatory structure. Sometimes the situation is not quite so simple. Sometimes knocking out a loop will cause the model to stop oscillating, but the loop won't oscillate on its own. In that case the oscillation is caused by at least two loops acting in concert (often an oscillatory loop and a destabilizing loop). Sometimes there will be more than one loop that by itself is oscillatory – the model will oscillate as long as any one of those oscillatory loops is active.

Going from model-insight to real-world-insight. Understanding the model is intellectually satisfying, but it doesn't do any good in the world. Ultimately, we aren't interested in the model; we are interested in controlling an actual social system.

After discovering the structure(s) responsible for model behavior, ask yourself if the real world possesses analogous structures. (Note: The real world probably *does* possess similar structure – after all, that's probably why you put the structure into the model in the first place). If the real world possesses analogous structure, that structure will *tend* to cause the world to behave in a way analogous to your model. Note, you are not saying that the real world produces observable behavior of that type – lots of things from lack of measurement to additional structure might cause real-world observations to differ. All you are saying is that the real world contains a structure that *could* produce such-and-such behavior. If the behavior is undesirable, think of what can be done in the real world

to eliminate or reduce the likelihood of that behavior. You will need to translate from your model context (e.g. reducing the *time-to-change-workforce* stabilizes the system) to a real-world context (e.g. we need to keep resumes on file, or we should consider hiring contract workers).

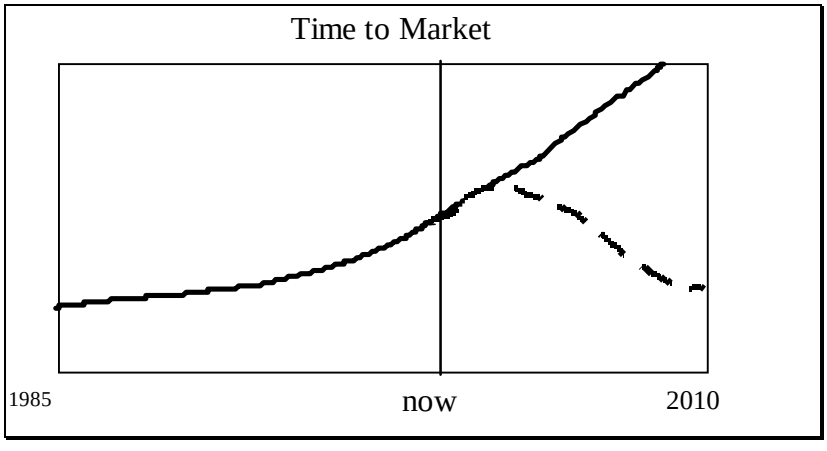
The value of simplicity and the need to understand the model *now*. The potential difficulty of understanding the causes of behavior in your model is why you want to keep your model as dynamically simple as possible. Note, *detail complexity* does not necessarily produce analytical problems; the real culprit is *dynamic complexity* –i.e. lots of loops. Keep in mind that you *need* to understand your model at *this* stage; do not put off understanding to a later time: Your model will never be easier to understand than it is now. The idea behind the standard method is that your understanding of the model will grow along with the model – so, start simple.

Presentation. You may choose to show relevant reference modes, loops, model structure, output, insights, and/or policy conclusions. The particular mix of these things in your presentation will depend on what you have discovered. If the modeling hasn't led to any insights, then you won't dwell very much on insights. If, on the other hand the modeling has led to many insights, then to give yourself time to discuss them all, you might need to be very brief in discussing the model.

Reference modes and mystery. Reference modes provide an excellent grounding for your audience, clueing them into what's important and what's unimportant. If you start with reference modes, you will not get as many off-the-wall suggestions (e.g. "Does your model include the kitchen sink?"). Further, you can motivate your presentation by describing reference modes in terms of a problem or a question (e.g. "Time-to-market has risen. No one knows for sure why."). People are intrigued by a good mystery, so reference modes presented in this way will cause folks to pay attention.

You should only present the reference modes that are important for understanding and motivating the particular hypothesis you want to discuss.

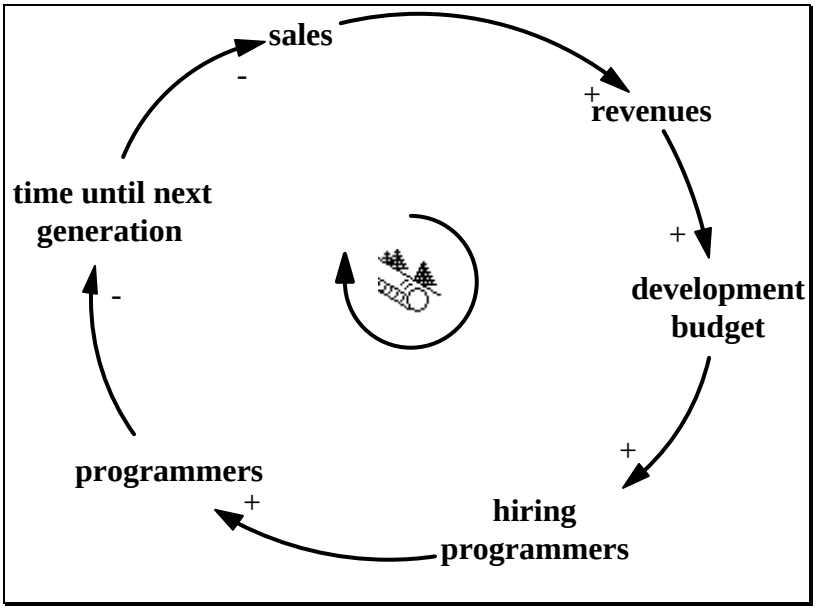
For example, the hypothesis I chose to model was the programmers as solution loop (see below). The relevant reference mode shows rising development times. With the reference mode projected on a screen, one might offer the oral commentary: "The key problem we're facing is that each new release of our product has been taking longer and longer. We are afraid that development times will continue to rise, but hope that, through our work



here, we can bring development times back down. There are processes that tend to lengthen the development process and there are processes that tend to shorten it. Clearly, the 'lengtheners' have swamped the 'shorteners'. We want to start our consideration of the problem with one of these "swamped" processes that in isolation would make the process go faster. We figure if we understand this mechanism, maybe we can strengthen it so that it can hold its own against the forces of evil."

Hypothesis. Your simulation model will probably be too complicated/detailed to be a clear explanation of your hypothesis, so you will probably want to present your hypothesis in words and loops.

For example my hypothesis is:



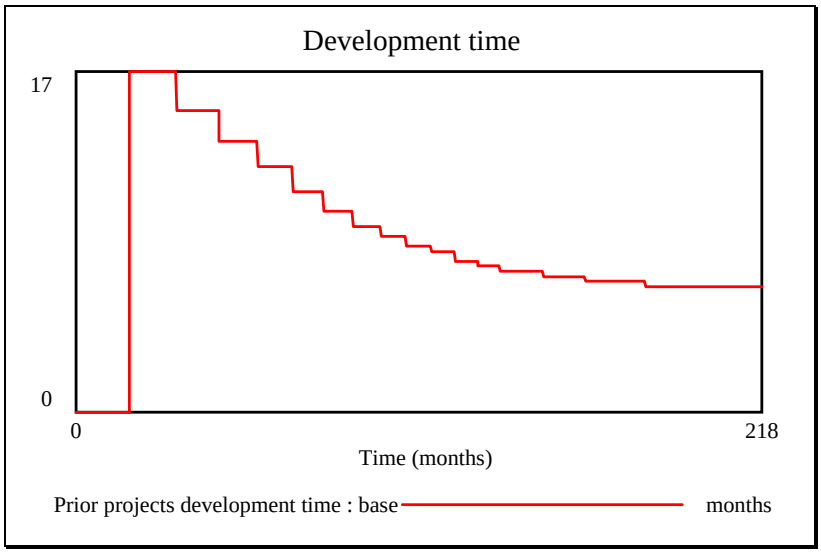
I would put up an overhead of this loop and trace the links to show how it works. I would end by referring to the reference mode: "It appears that this loop could cause ever-declining product development times". Note, there is no need to put up any of the other hypotheses.

Model. You may have time to present your entire model in stock and flow terms; if not just present the most important parts.

If you are presenting the model to a client, you need to be prepared for the client to disagree with the way you have represented the company. If you have interesting output, you may find yourself in a tough position: The client has just called into question some structure in your model and the most interesting part of the talk, which is yet to come, is based on the model being O.K. You can head this off if you quickly show the output *before* you describe the model. Showing the output first, will again focus your audience on what's important (behavior and, yes, structure, too, but only as it affects behavior) so suggestions to improve the structure will tend to be more relevant. Furthermore, any

surprising behavior will intrigue your audience so they will want to hurry through the model so they can understand what causes the interesting output; they will be less willing to take the time to make minor points about the structure. Finally, the reason people will often raise questions about model structure is that they are afraid of what the model *might* show; they combat this unfocussed fear by tearing down the model. The output is usually more interesting and less frightening than their fears, so seeing the output first will relax them.

For example, The model I built of the above hypothesis has interesting output. Just before presenting the model, I would be wise to put up some of this output,



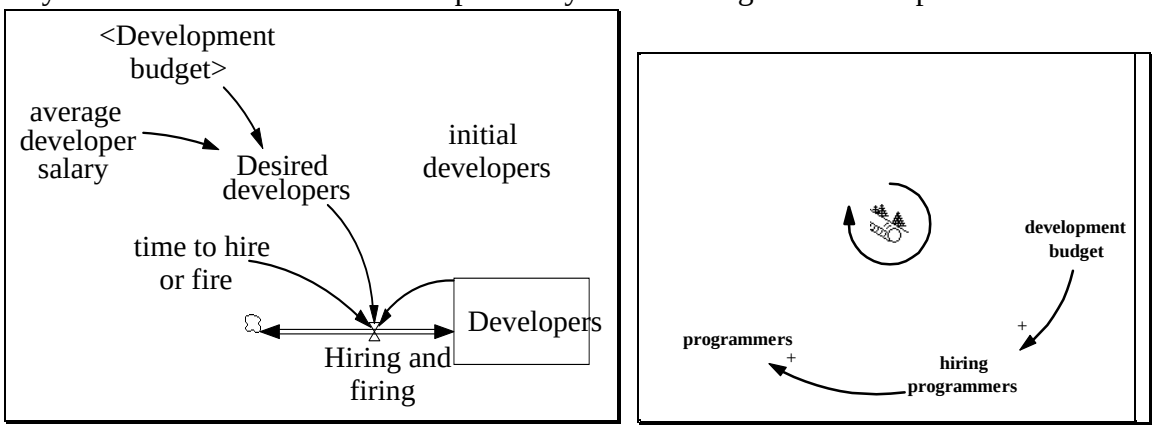
with the explanation: "This chart shows declining project development time. The stair-step pattern reflects the fact that we are tracking the development time of the *most recently completed* project – so, development time is fixed, until the next project is completed. Interestingly, The development-time stagnates eventually. The loop mechanism, which we thought would cause development time to fall continually, somehow runs out of steam. I want to spend most of our time going over the reasons that the simulation model produces this behavior, but first let me show you what is in the model". *[NOTE: The stair-step pattern is unusual output for a system dynamics model. Most SD models have gradual, continuous actions, rather than discrete, all-at-once actions. The stair steps are <u>not</u> a strength of this model.]*

Now, you are in a position to show the model structure. You should break the model into digestible pieces so that the overheads of the model are simple. (Your modeling should also be done in this way, using multiple views in Vensim). One organizing strategy for your overheads is to move backwards in the causal chain. The challenge here is to keep your audience mindful of where they are in the loop, so that they see how the big-picture loop has been translated into the more operational stock-and-flow structures.

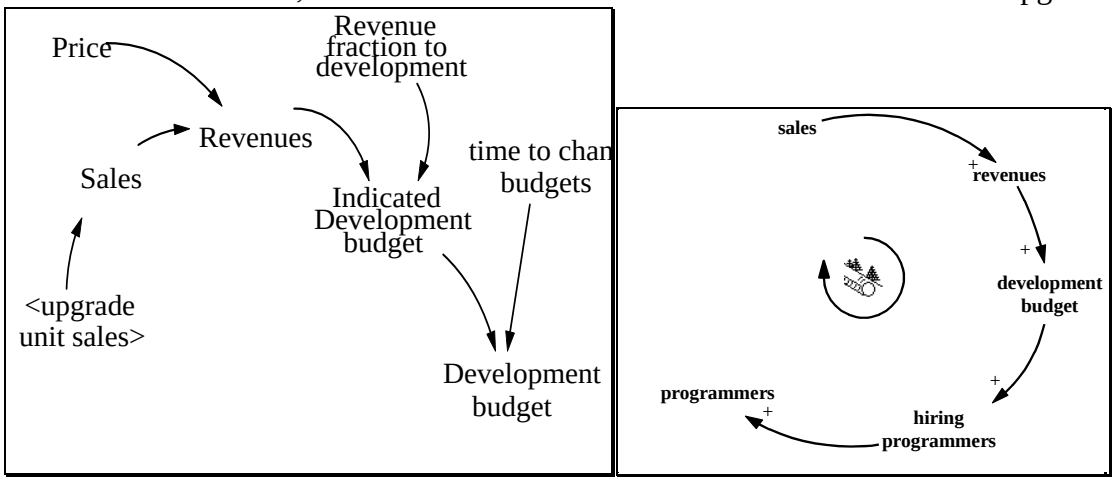
For example, you might present a model of the above hypothesis in the following way (while periodically showing the loop diagram in a "you are here" way)

You might begin:

"The number of developers is represented as a stock that fills up with the hiring rate, which closes the gap between the number of developers we actually have and the number of developers hat we can afford. To figure out how many developers we can afford we only need to know the usual developer salary and the budget for developers".

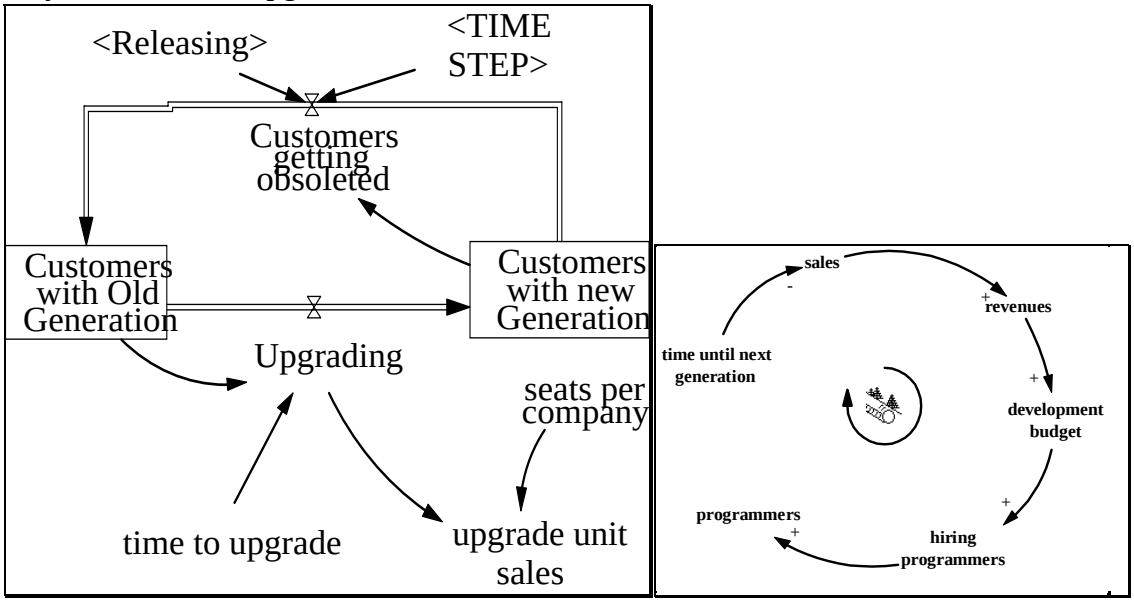


"The budget for developers changes gradually, but is basically a constant fraction of revenues. Revenues are simply unit sales multiplied by the price. In our loop diagram, we weren't very clear about where sales come from. I originally had in mind an increasing market share as the functionality of the product improved, but as I was putting this together, I realized there were other components of sales as well, including sales from upgrades of the product. Increasing market share may be important, but what I decided to focus on for the moment was upgrades – It seemed to me that the main reason I buy another version of Microsoft Word is not because the functionality is suddenly better than Word Perfect, but because a new release has come out and I want to upgrade."

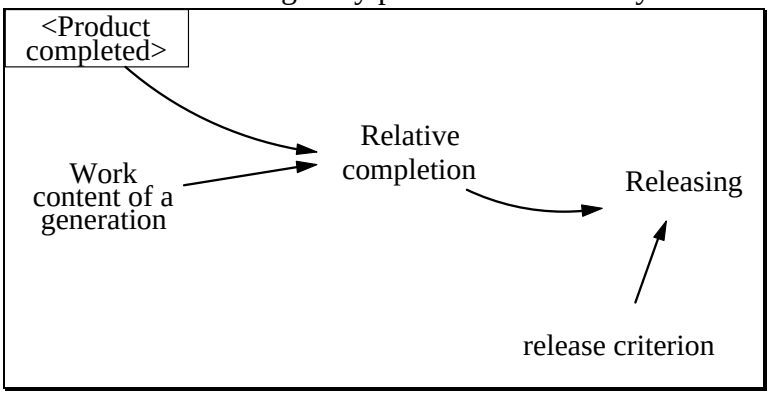


"The next diagram shows how customers upgrade. When a product is released, all existing customers are 'obsoleted'. Customers then start buying the new version of the

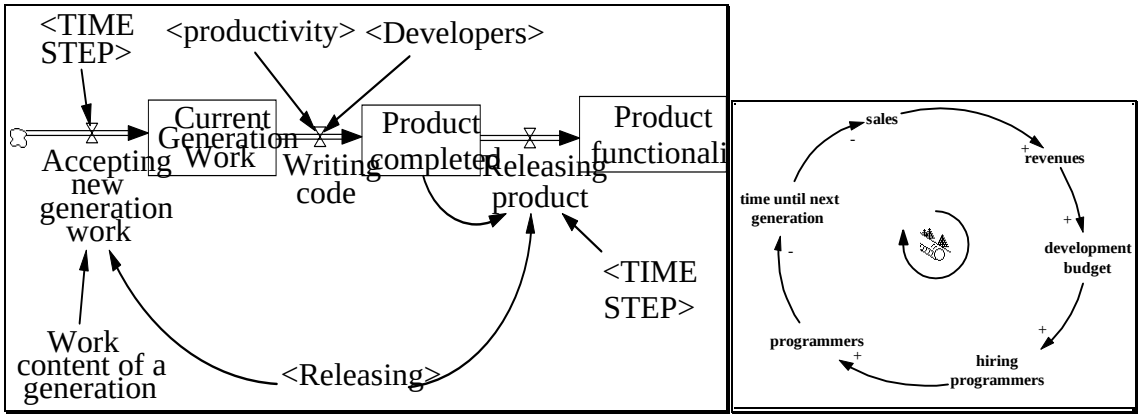
software. Most of our customers are companies who have a number of licenses, so for every customer that upgrades, we sell some number of 'boxes' ".



"We release a new product when we have completed it. Usually, we decide to release it without all of the originally planned functionality; we simply carry over the un-included functionality into the *next* release. Our "release criterion" historically has been when we have completed about 95% of the originally planned functionality."



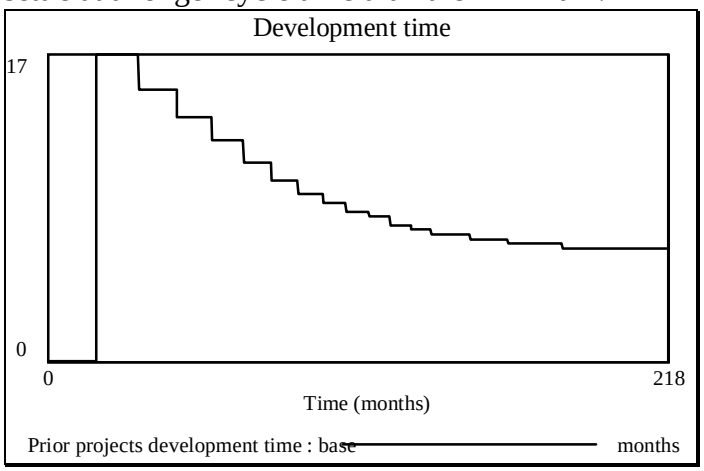
"Product completed, but not released is a stock. When the product is released, our product functionality goes up. Also when a product is released we accept a new slug of functionality to start working on for the next generation. Programmers work on the product to complete it. And, of course, programmers come from our first slide showing the hiring process."



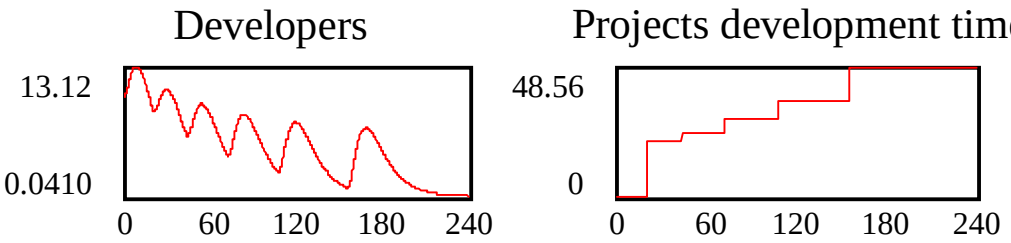
Output. You should show some output. If the model behaves exactly as you anticipated, you don't need to dwell on the output. Usually, though there will be at least one or two items that you (and your client) didn't anticipate.

For example:

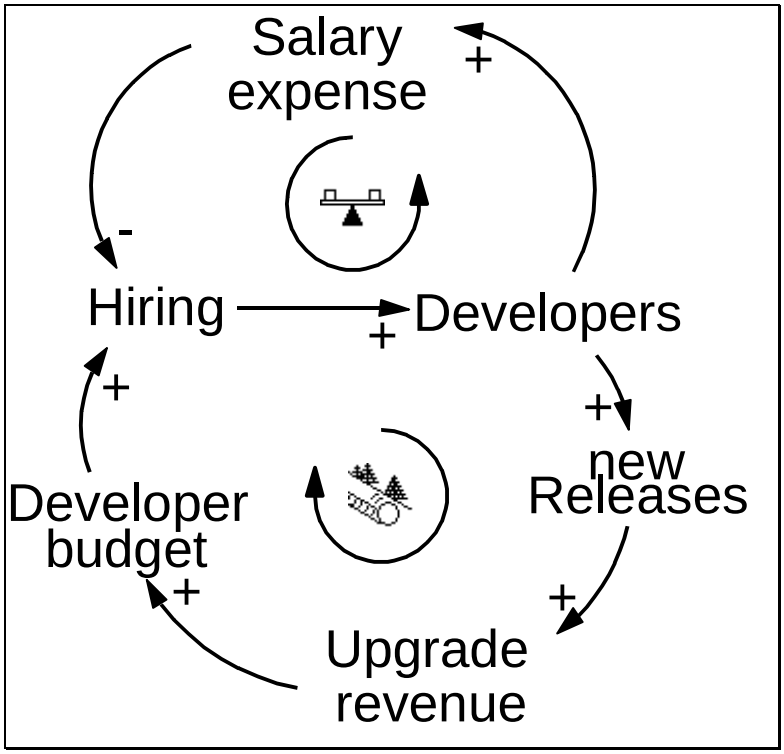
"The time it takes to develop a product, flattens out and stops declining. This is because our income depends – in this little structure – on the frequency of our releases. Let's say we release as often as possible. This implies a maximum revenue from releases. This maximum revenue then determines the maximum number of programmers we can afford. Of course, we may never reach that maximum revenue, if the maximum revenue does not let us afford the as many programmers as we need to achieve minimum cycle time. In this case, we will settle at a longer cycle time than the minimum.



You may want to provide your explanations both in managerial terms (as above) and with loops. Your classmates won't have trouble following a causal loop diagram. Your clients may have a bit more difficulty, but they may also find "thinking in loops" is neat. For example, another insight is that if the programmers are not productive enough to cover their own salaries, the developers will decay away (while oscillating) and development times will lengthen. As the following output shows.



To support the above explanation, you might show the following diagram with the comment that the behavior will depend on which loop dominates. That is, the behavior will depend on whether each new worker creates more budget or more expense.



Insights and conclusions. You should present each insight as soon as you've given your audience the information necessary to understand it. So, for example, after explaining the first surprising behavior (there is a minimum development time (maximum revenue) even in the best of circumstances) you could tick off a number of insights:

- There is a maximum revenue that is achievable from upgrades alone
- The maximum revenue from upgrades is limited by the time it takes for customers to buy again
- This amount of time is determined by development time, but also by selling time (how long does it take for customers to upgrade after a new release?)
- The company should consider factors affecting selling time as well as factors affecting development time.

- It is unlikely that upgrades alone will produce enough cash to achieve a minimum cycle time
- If reducing the cycle time requires more programmers, they will be supported from new sales, that is from gains in market share or growth in the market overall. Unfortunately, this source of revenue is not sustainable. Only upgrades is a sustainable revenue source.

Insights from the second aspect of surprising behavior (the company will decay away if each programmer can't support himself) would include:

- Salary must be less than the budget generated by each new employee.
- The budget generated by each employee depends on productivity, seats per customer, functionality in a release (i.e. work content), fraction of revenues to budget, and price.
- In the effort to reduce selling time, you may move budget out of development and into advertising. But, make sure the fraction of the budget to development does not fall below a critical level.
- Reducing functionality per release (or raising price, or increasing seats per customer) permits a lower budget fraction to development.

At the end of your talk you may want to provide a slide listing all of the insights and conclusions.

Equations. In large client meetings you probably will not want to show equations – too many in the audience will not benefit from them and, if your purpose is to reveal behavior and an insight or two, they may distract your audience from the path you want to follow. If an equation is interesting or important, by all means put it up – but only if it contributes to your overall message, not if it only shows how smart you are. (Your client already knows you are smart).

For your information the views above are the stock and flow counterpart to the following equations. Note, most of the formatting was done with Vensim's document tool. The tool (like most tools) can be configured by right clicking on the button. (Choose to *display* equations, text, and units. Under *multiple equations* choose *all.* Under "ordering", choose *alphabetic by group.* Under "*formatting"* choose *terse.* The separate sections were created with Vensim's *group* facility. You can assign an individual equation to a particular group (and create a new group) in the equation editor by using the *group* drop-down list in the lower left corner. You can assign an entire view to a group by selecting the entire view (ctrl-A), and then choosing the menu item: View>>Act on Selected>>Group. (Note that an equation can be assigned to only one group. So you probably don't want to assign shadow variables in a view to the view's group).

```
********************************
  .Developers
********************************
average developer salary = 5000
        Units: $/(month*person)
Desired developers = Development budget / average developer salary
        Units: people
Developers = INTEG( Hiring and firing , initial developers )
        Units: people
Hiring and firing = ( Desired developers - Developers ) / time to hire or fire
        Units: people/month
initial developers = 10
        Units: people
time to hire or fire = 6
        Units: months

********************************
  .Financial
********************************
Development budget =
   smoothi ( Indicated Development budget , time to change budgets,
             initial developers * average developer salary )
        Units: $/month
Indicated Development budget = Revenues * Revenue fraction to development
        Units: $/month
Price = 1000
        Units: $/box
Revenue fraction to development = 0.25
        Units: fraction
Revenues = Sales * Price
        Units: $/month
Sales = upgrade unit sales
        Units: boxes/month
time to change budgets = 12
        Units: months

********************************
  .Product development
********************************
Accepting new generation work =
    if then else ( Releasing = 1, Work content of a generation / TIME STEP , 0)
        Units: functions/month
Current Generation Work =
  INTEG( Accepting new generation work - Writing code, Work content of a generation
  )
        Units: functions
```

Product completed = INTEG(Writing code - Releasing product , 0)
Units: functions
Product functionality = INTEG(Releasing product , 10)
Units: functions
Releasing product =
if then else (Releasing = 1, Product completed / TIME STEP , 0)
Units: functions/month
Writing code = Developers * productivity
Units: functions/month

****
.Productivity
****
maximum productivity = 0.05
Units: functions/(month*person)
productivity = maximum productivity * work availability effect on productivity
Units: functions/(month*person)
relative work remaining = Current Generation Work / Work content of a generation
Units: fraction
work availability effect on productivity =
work availability effect on productivity f(relative work remaining)
Units: dmnl
work availability effect on productivity f ([(0,0)-(10,1)],(0,0),(0.060423,0.324561)
,(0.148036,0.609649),(0.392749,0.938596),(0.770393,0.991228),(1,1)
,(2,1),(10,1))
Units: dmnl

****
.Release criterion
****
Relative completion = Product completed / Work content of a generation
Units: fraction
release criterion = 0.95
Units: fraction
Releasing = if then else (Relative completion >= release criterion , 1, 0)
Units: dmnl
Work content of a generation = 10
Units: functions

****
.Upgrading
****
Customers getting obsoleted =
if then else (Releasing = 1, Customers with new Generation/ TIME STEP , 0)
Units: companies/month

NOTE: If-then-else equations are usually POOR modeling practice. Making an equation explicitly depend on the time-step is also usually poor modeling practice. If we were smarter we would figure out how to have a continuous formulation that did not depend explicitly on time.

Customers with new Generation =
INTEG(Upgrading - Customers getting obsoleted, 0)
Units: companies

Customers with Old Generation =
INTEG(Customers getting obsoleted - Upgrading, 1000)
Units: companies

seats per company = 5
Units: boxes/company

time to upgrade = 4
Units: months

upgrade unit sales = Upgrading * seats per company
Units: boxes/month

Upgrading = Customers with Old Generation / time to upgrade
Units: companies/month

****
.Statistics
****

clearing monitor for next project =
if then else (Releasing = 1, Current project's development time / TIME STEP , 0)
Units: months/month

clearing record for next project =
if then else (Releasing = 1, Prior projects development time / TIME STEP , 0)
Units: months/month

Current project's development time =
INTEG(time passing on current project- clearing monitor for next project , 0)
Units: months

Prior projects development time =
INTEG(recording project development time- clearing record for next project , 0)
Units: months

recording project development time =
if then else (Releasing = 1, Current project's development time / TIME STEP , 0)
Units: months/month

time passing on current project = 1
Units: months/month

```
********************************
  .Control
********************************
FINAL TIME = 120
        Units: months
INITIAL TIME = 0
        Units: months
SAVEPER = TIME STEP
        Units: months
TIME STEP = 0.25
        Units: months
```