

Appendix 3-3- The complete model formulation for detailed multiple release software product simulation model

In this appendix the model formulations for the detailed simulation model (discussed in appendix 3-1) are documented. This model was produced as part of the research project leading to the paper “Dynamics of Platform-based Product Development”. The model, however, was too complex to be discussed in the paper. Appendix 3-1 elaborates on the role of this model in the research process and discusses some of the testing and analysis tools developed for working with this model.

The model is broken into different views, each focusing on a different aspect of the development process and other related dynamics. Each view is followed by the equations of the variables introduced in that view, sorted alphabetically. At the end of the document, all the variables of the model are sorted alphabetically and their group/view is listed. Therefore you can find where to look up the equations of any variable by looking up the group name at the end of the document and going to the equations for the relevant group. This should prove most useful for tracking the shadow variables in each view. The color codes used in these views are explained in appendix 3-1.

The model includes design, development, testing, defects in the field, bug fixing, underlying code base quality, scheduling, and sales and financial metrics of 6 consecutive releases of a software product. Several major feedback structures relevant to quality and delays in software projects are captured in the model. The main ones include: resource sharing between different releases and current engineering, effects of pressure on productivity and error rate; code base and architecture quality on error rate and design quality; architecture complexity on productivity; bug fixing on scope; re-planning on re-plan threshold; quality, delay, and features on sales; profit on resources; and endogenous start of different releases and endogenous sales.

At least three pieces of formulation in this model offer relatively new and potentially useful options for modeling some common problems and therefore may merit closer attention. These pieces are briefly mentioned below and the references to the relevant model sector are provided.

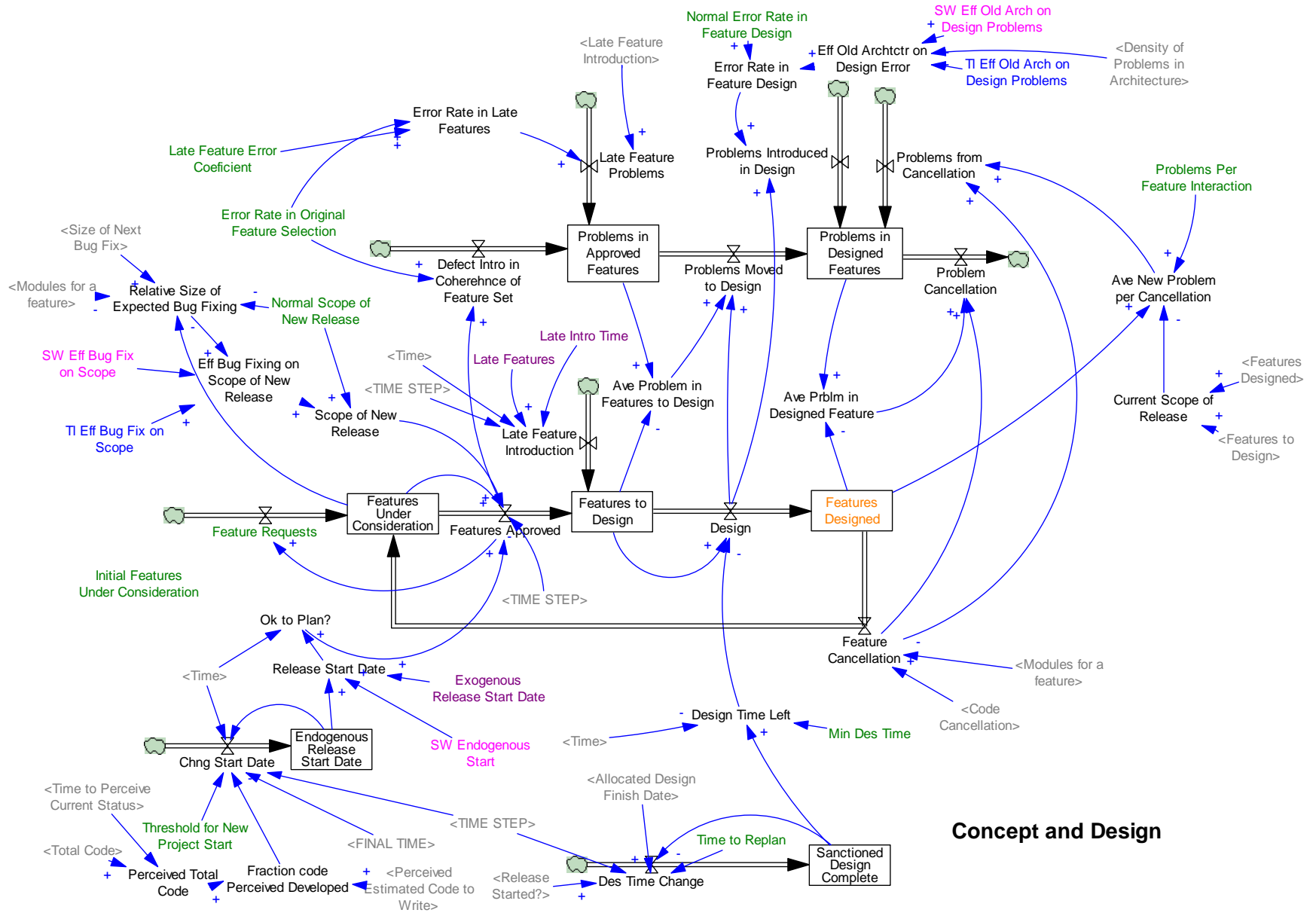
Coflow of tasks and errors- Historically project dynamics’ formulations of tasks and errors either divide tasks into correct/flawed categories, or use a coflow of “task quality” (changing

between 0 and 1) or “changes” (required changes accompanying the flow of tasks) that goes with tasks. In the current model I have used an alternative formulation that tracks the number of flaws in the tasks developed, and uses the density of flaws in a consistent formulation to determine chances of discovery of problems in quality assurance phases. This formulation is most useful to follow the logic of many projects in which problems are conceptually different from tasks. For example in the software development process a module of code (a task) may have several flaws (bugs) that can be discovered (become an MR: Modification Request) or not, and the density of these problems determines the chances of their discovery in different testing phases. Details of this formulation can be seen in “Development” and “Test” views of the model.

Tracking the problems in the field- As products are sold to customers, the bugs inside the products also flow into different customer sites. These bugs can be independently found by different customers, reported to the company, fixed, and patched for all, or part of the existing customer set. The coflow structure to capture these possibilities is non-trivial because each error is flowing in all of the sites, but can be found in any of them. Therefore the enclosed formulation may prove useful for other researchers who tackle similar modeling problems. “Defects in the Field” view of the model elaborates on this formulation.

Resource anticipation between different releases with different priorities- In allocating resources between different overlapping releases of a software, the project managers for each release should perceive what the priority of their release is (depending on where in the queue of projects it is located) and plan their schedules and expected resources accordingly. While in logic this is simple, the implementation of the formulation in SD software is not without challenges because as different releases start and finish, their priority changes, and their perceived priority should adjust according to their current position in project priority queue. The formulations on the “Scheduling” view of the model offer one solution to this modeling challenge.

This model includes 448 symbols, 17 lookups, 220 Levels, 1151 Auxiliaries, and 147 constants (Total of 1519 variables).



Concept and Design

Ave New Problem per Cancellation[Release] = ZIDZ (Problems Per Feature Interaction * Features Designed[Release] , Current Scope of Release[Release])

Units: Problem/feature

This equation assumes that cancellations have detrimental effects to the extent that they interact with other features in the code. Therefore the problems they create also correlates with fraction of features inside the code designed. However, for the part of code not designed, the cancellations create no new issues, since those pieces can be designed with knowledge of cancellation.

Ave Prblm in Designed Feature[Release] = ZIDZ (Problems in Designed Features[Release] , Features Designed[Release])

Units: Problem/feature

Average number of design problems in designed features.

Ave Problem in Features to Design[Release] = ZIDZ (Problems in Approved Features[Release] , Features to Design[Release])

Units: Problem/feature

Average number of (coherence) problems in features to be designed.

Chng Start Date[R1] = 0

Chng Start Date[Later Releases] = if then else (Endogenous Release Start Date[Later Releases] = FINAL TIME , if then else (VECTOR ELM MAP (Fraction code Perceived Developed[R1] , Later Releases - 2) > Threshold for New Project Start , 1, 0) * (Time + 1 - Endogenous Release Start Date[Later Releases]) / TIME STEP , 0)

Units: Dmnl

This equation puts the release start date for later projects (not first release) at the TIME+1, as soon as the project before them is completed to a Threshold level in development phase.

Current Scope of Release[Release] = Features Designed[Release] + Features to Design[Release]

Units: feature

Defect Intro in Coherence of Feature Set[Release] = Error Rate in Original Feature Selection[

Release] * Features Approved[Release]

Units: Problem/Month

Des Time Change[Release] = if then else ("Release Started?"[Release] = 1 :AND: Sanctioned Design Complete[Release] = 0, Allocated Design Finish Date[Release] / TIME STEP , "Release Started?"[Release] * (Allocated Design Finish Date[Release] - Sanctioned Design Complete[Release]) / Time to Replan)

Units: Dmnl

We set new desired times for finishing the design based on what we get allocated.

Design[Release] = Features to Design[Release] / Design Time Left[Release]

Units: feature/Month

The development of requirements is enforced to finish by the time that designs are scheduled.

Design Time Left[Release] = Max (Min Des Time , Sanctioned Design Complete[Release] - Time)

Units: Month

Design time left depends on the scheduled time for design and a minimum time needed to make any design changes.

Eff Bug Fixing on Scope of New Release[Release] = TI Eff Bug Fix on Scope (Relative Size of Expected Bug Fixing[Release]) * SW Eff Bug Fix on Scope + 1 - SW Eff Bug Fix on Scope

Units: Dmnl

The effect of bug fixing on scope of new release depends on how much bug fixing is there compared to what we are planning to accomplish in the next release, if there was no bugs.

Eff Old Archctr on Design Error = TI Eff Old Arch on Design Problems (Density of Problems in Architecture) * SW Eff Old Arch on Design Problems + 1 - SW Eff Old Arch on Design Problems

Units: Dmnl

The effect of quality of old design and architecture on how good a job we do in design of new pieces.

Endogenous Release Start Date[R1] = INTEG(Chng Start Date[R1] , 0)

Endogenous Release Start Date[Later Releases] = INTEG(Chng Start Date[Later Releases] , FINAL TIME)

Units: Month

This release start date is set to start a release as soon as the former one passes a threshold in % of code developed.

Error Rate in Feature Design[Release] = Normal Error Rate in Feature Design[Release] * Eff Old Archctr on Design Error

Units: Problem/feature

The error rate in designing features. Includes low quality requirements and unclear specifications etc. Depends on the quality of old code and architecture..

Error Rate in Late Features[Release] = Error Rate in Original Feature Selection[Release] * Late Feature Error Coefficient

Units: Problem/feature

The error rate in the design of late features.

Error Rate in Original Feature Selection[Release] = 0.1

Units: Problem/feature

Rate of problems in coherence of combinations chosen for new features.

Exogenous Release Start Date[Release] = 0, 6, 12, 18, 24, 30

Units: Month

Release Start Dates, as an exogenous parameter

Feature Cancellation[Release] = Code Cancellation[Release] / Modules for a feature

Units: feature/Month

The speed of cancellation of features already designed, under time pressure.

Feature Requests = SUM (Features Approved[Release!])

Units: feature/Month

The rate of request for new features, mainly from customer needs and market pressure. For the equilibrium, it can be put at the features approved level so that there is no net change in the total number of features under consideration (assuming no cancellation)

Features Approved[Release] = Features Under Consideration * Scope of New Release[Release] / TIME STEP * "Ok to Plan?"[Release]

Units: feature/Month

At the time of OK to Plan, a fraction of total features under consideration are finalized to be designed.

Features Designed[Release] = INTEG(Design[Release] - Feature Cancellation[Release] , 0)

Units: feature

Number of features designed and ready to be coded

Features to Design[Release] = INTEG(- Design[Release] + Features Approved[Release] + Late Feature Introduction[Release] , 0)

Units: feature

Number of features to be designed before being coded.

Features Under Consideration = INTEG(Feature Requests - SUM (Features Approved[Release!] - Feature Cancellation[Release!]) , Initial Features Under Consideration)

Units: feature

The number of features under consideration for future releases.

Fraction code Perceived Developed[Release] = 1 - XIDZ (Perceived Estimated Code to Write[Release] , Perceived Total Code[Release] , 1)

Units: Dmnl

The fraction of code for each release that is perceived to be left for development.

Initial Features Under Consideration = 2000

Units: feature [0,12000] The constant for the initial number of features under consideration for each release.

Late Feature Error Coefficient = 3

Units: Dmnl

How many times of the normal feature additions do we encounter problems with the design of late features.

Late Feature Introduction[Release] = if then else (Late Intro Time[Release] = Time , Late Features[Release] / TIME STEP , 0)

Units: feature/Month

The rate of introduction of late features. Controlled as an exogenous shock here.

Late Feature Problems[Release] = Error Rate in Late Features[Release] * Late Feature Introduction[Release]

Units: Problem/Month

Rate of introduction of problems into approved features through introduction of late features.

Late Features[Release] = 0

Units: feature

The number of new features introduced at some point in the process

Late Intro Time[Release] = 15

Units: Month

Time at which the late features are introduced

Min Des Time = 0.25

Units: Month

Minimum time needed to finish the designs, even if after schedule.

Normal Error Rate in Feature Design[Release] = 0.1

Units: Problem/feature

Normal Error Rate in Feature Design

Normal Scope of New Release[Release] = 0.2

Units: Dmnl

The normal scope of the projects, as a fraction of all the features under consideration, if there was no bug fixing.

"Ok to Plan?"[Release] = if then else (Time = Release Start Date[Release] , 1, 0)

Units: Dmnl

A switch which equals 1 at the time step when we decide to start a new release.

Perceived Total Code[Release] = DELAY1 (Total Code[Release] , Time to Perceive Current Status)

Units: module

The perceived total code for the release.

Problem Cancellation[Release] = Feature Cancellation[Release] * Ave Prblm in Designed Feature[Release]

Units: Problem/Month

The problems are removed along with cancelled features.

Problems from Cancellation[Release] = Feature Cancellation[Release] * Ave New Problem per Cancellation[Release]

Units: Problem/Month

The feature cancellation can result in addition of design problems for the rest of the code, since pieces are removed from otherwise coherent architecture.

Problems in Approved Features[Release] = INTEG(Defect Intro in Coherence of Feature Set[Release] - Problems Moved to Design[Release] + Late Feature Problems[Release] , Features to Design[Release] * Error Rate in Original Feature Selection[Release])

Units: Problem

Problems in Designed Features[Release] = INTEG(Problems Moved to Design[Release] + Problems from Cancellation[Release] - Problem Cancellation[Release] + Problems Introduced in Design[Release] , Features Designed[Release] * Error Rate in Original Feature Selection[Release])

Units: Problem

Problems Introduced in Design[Release] = Design[Release] * Error Rate in Feature Design[Release]

Units: Problem/Month

The rate of introduction of new problems depends on the design speed as well as the quality of design.

Problems Moved to Design[Release] = Design[Release] * Ave Problem in Features to Design[Release]

Units: Problem/Month

Coherence problems carry forward into designed features.

Problems Per Feature Interaction = 0

Units: Problem/feature

Number of problems arising as a result of each feature interaction.

Relative Size of Expected Bug Fixing[Release] = XIDZ (Size of Next Bug Fix , Features Under Consideration * Normal Scope of New Release[Release] * Modules for a feature , 3)

Units: Dmnl

This tells how big will the bug fixes be in the upcoming release, compared to the desired number of features.

Release Start Date[Release] = SW Endogenous Start * Endogenous Release Start Date[Release] + (1 - SW Endogenous Start) * Exogenous Release Start Date[Release]

Units: Month

Release start date can be exogenous or endogenous.

Sanctioned Design Complete[Release] = INTEG(Des Time Change[Release] , 0)

Units: Month

This is the variable which drives time pressure to design stage.

Scope of New Release[Release] = Normal Scope of New Release[Release] * Eff Bug Fixing on Scope of New Release[Release]

Units: Dmnl

The fraction of total features under consideration that we decide to include in this release.

SW Eff Bug Fix on Scope = 1

Units: Dmnl

Put 1 to adjust the scope of next release according to the level of bug fixing planned for it. Put 0 not to have this effect.

SW Eff Old Arch on Design Problems = 1

Units: Dmnl

Put 1 to have the effect of old architectural and design problems on quality of current design active.

SW Endogenous Start = 1

Units: Dmnl

Put 1 to have the starts of release endogenously determined, put 0 for exogenous starts.

Threshold for New Project Start = 0.6

Units: Dmnl

Threshod for starting a new release. As soon as the current release passes this threshold, the new release will be started.

Time to Replan = 1

Units: Month

This is the time constant for adjustment of expectations about release date, code complete etc, to the data showing that it is going up or down.

TI Eff Bug Fix on Scope ([(0,0)-(4,1)],(0,1),(0.452599,0.692982),(1,0.4),(1.3945,0.232456) ,(2,0.1),(2.77676,0.0263158),(4,0))

Units: Dmnl

Effect of bug fixing scope on the scope of the next release

TI Eff Old Arch on Design Problems ([(0,0)-(1,2)],(0,1),(0.269113,1.16667)
,(0.40367,1.32456),(0.562691,1.60526),(0.691131,1.80702),(0.82263,1.9386) ,(1,2))

Units: Dmnl

Table for effect of architecture and code base quality on the quality of design

Development

Allocated Design Time Left[Release] = Estmtd Design Time Left[Release] * Prcnt
Time Allocated[Release]

Units: Month
allocated design time left.

Ave Dfct Dnsty in Acptd Code[Release] = ZIDZ (Defects in Accepted Code[Release]
, Code Accepted[Release])

Units: Defect/module
Defect density in the accepted code is an important quality measure for us.

Ave Dfct in Rwrk Code[Release] = ZIDZ (Defects in MRs[Release] , Code Waiting
Rework[Release])

Units: Defect/module
The number of defects that can be fixed by fixing the MR.

Ave Dfct in Tstd Code[Release] = Code Covered by Test * Dfct Dnsty in Untstd Code[
Release]

Units: Defect/Test
The average number of defects that one can expect to find in a test, given the average
density of defects.

Bug Fixes Designed[Release] = Code to Fix Bugs[Release]

Units: module/Month
The bug fixes included for the upcoming release.

Code Accepted[Release] = INTEG(Test Acceptance[Release] , 0)

Units: module

Code Cancellation[Release] = Code to Develop[Release] * Frctn Code Cancellation[
Release]

Units: module/Month
Under pressure we can cancel some of the codes that are designed to be written. This can
be both intentionally, or unintentional.

Code Covered by Test = 1

Units: module/Test This should be a function of test coverage

Code Developed[Release] = INTEG(Development[Release] + Rework[Release] - Test
Acceptance[Release] - Test Rejection[Release] , 0)

Units: module
The amount of code developed, but yet not successfully tested.

Code Passing Test[Release] = Totl Code to Tst[Release] * Frctn Tst Running[Release]
Units: module/Month

The rate of code testing is assumed to be proportional to the amount of test that is performed out of total tests to be performed. This assumption is OK for products which run only the related fraction of the tests.

Code to Develop[Release] = INTEG(Design Rate[Release] - Development[Release] - Code Cancellation[Release] + Bug Fixes Designed[Release] , 0)
Units: module

We start from no new code to develop, since no requirement is written at the beginning.

Code Waiting Rework[Release] = INTEG(Test Rejection[Release] - Rework[Release] , 0)

Units: module
The total amount of code waiting rework.

Current Dev Schedule Pressure[Release] = ZIDZ (Estmtd Dev Time Left[Release ,Current] , Development Time Left[Release])
Units: Dmnl

The schedule pressure depends on how long we have left for the development phase vs. how long we estimate this phase to take. Note that this schedule pressure is based on observed productivity, rather than the normal/base productivity. Therefore it is anchored on what is culturally considered right productivity in the organization not an absolute level of productivity. Therefore it should not be used for determining productivity levels.

Defects Accepted[Release] = (Tstd Code Acptd[Release] * Dfct Dnsty in Accepted Code[Release] + (Test Acceptance[Release] - Tstd Code Acptd[Release]) * Dfct Dnsty in Untstd Code[Release])
Units: Defect/Month

The defects can be either in the part tested, or in the part not tested. The density of defect in the untested part is same as average untested code, while the density in the tested part is calculated based on Poisson distribution of defects.

Defects Fixed[Release] = Rework[Release] * Ave Dfct in Rwrk Code[Release] * Frac Errors Found in Rwrk
Units: Defect/Month

It is assumed that a fraction of the defects in the reworked code are removed, and a fraction remain undiscovered.

Defects found[Release] = Test Rejection[Release] * Dfct Dnsty in Rejctd Code[Release]
Units: Defect/Month

Defects are carried with the code, the higher the rejection rate and the density of defects in the code rejected, the higher will be the defects that are found.

Defects from Dev[Release] = Development[Release] * Error Rate New Dev[Release]
Units: Defect/Month

The defects that come from development depend on the error rate and the rate of development we do.

Defects Generated[Release] = Defects from Dev[Release] + Dfct from Rwrk[Release]
Units: Defect/Month

The total defect generated is a function of both defects in the new code as well as in the old code.

Defects in Accepted Code[Release] = INTEG(Defects Accepted[Release] , 0)
Units: Defect

The number of defects in the accepted code.

Defects in Code Developed[Release] = INTEG(Defects Generated[Release] - Defects Accepted[Release] - Defects found[Release] + Dfcts Left in RWrk[Release] , 0)
Units: Defect

The defects from the code of last release are carried over to the current release.

Defects in MRs[Release] = INTEG(Defects found[Release] - Defects Fixed[Release] - Dfcts Left in RWrk[Release] , 0)
Units: Defect

We find defects through testing and deplete them by either submitting reworked code that has defects or by fixing them.

Design Rate[Release] = Design[Release] * Modules for a feature
Units: module/Month

The total number of modules to be developed depend on the modules in one feature, and the total number of features to be developed.

Desired Development Rate[Release] = Code to Develop[Release] / Development Time Left[Release]
Units: module/Month

The development rate needed to finish the coding on time for code complete

Dev Reschedule[Release] = if then else (Allocated Dev Finish Date[Release] > Time :AND: Sanctioned Code Complete[Release] = 0, Allocated Dev Finish Date[Release] / TIME STEP , if then else (Allocated Dev Finish Date[Release] > Time , 1, 0) * (Allocated Dev Finish Date[Release] - Sanctioned Code Complete[Release]) / Time to Replan)

Units: Dmnl

The (re)scheduling of the code complete date.

Dev Resources to New Dev[Release] = Min (ZIDZ (Dsrđ New Dev Resources[Release] , Dsrđ Dev Resources[Release]) * Development Resources to Current Release[Release] , Dsrđ New Dev Resources[Release])

Units: Person

The resources between rework and new development are distributed based on how much need there is for each, proportionately.

Dev Resources to Rework[Release] = Min (ZIDZ (Dsrđ Rwrk Resources[Release] , Dsrđ Dev Resources[Release]) * Development Resources to Current Release[Release] , Dsrđ Rwrk Resources[Release])

Units: Person

The resources between rework and new development are distributed based on how much need there is for each, proportionately.

Development[Release] = Feasible New Dev Rate[Release]

Units: module/Month

The development rate is based on resources allocated for development.

Development Time Left[Release] = Max (Min Dev Time , Sanctioned Code Complete[Release] - Allocated Design Time Left[Release] * (1 - "Frctn Des-Dev Overlap") - Time)

Units: Month

If we run over the code scheduled date, we can't push for faster than a minimum finish time for development.

Dfct Dnsty in Accepted Code[Release] = EXP (- Tst Effectiveness * Ave Dfct in Tstd Code[Release]) * Ave Dfct in Tstd Code[Release] * (1 - Tst Effectiveness)

Units: Defect/module

The defect density in accepted code is based on Poisson distribution of errors across code, leading to a chance of not discovering any defects for each test that has n errors, therefore we can calculate the expected error per accepted test.

Dfct Dnsty in Rejctd Code[Release] = Max (0, ZIDZ (Totl Code Tstd[Release]) * (Dfct Dnsty in Untstd Code[Release] - Dfct Dnsty in Accepted Code[Release]) + Test Rejection[Release] * Dfct Dnsty in Accepted Code[Release] , Test Rejection[Release]))

Units: Defect/module

We can't change the density of error in the untested code based by simply testing it. Therefore the density of defect in the exiting code (accepted or rejected) should be the same as that in the stock of code. This equation is based on the free mixing of errors in the stocks of code.

Dfct Dnsty in Untstd Code[Release] = ZIDZ (Defects in Code Developed[Release] , Code Developed[Release])

Units: Defect/module

The density of Defects in the code developed but not tested.

Dfct from Rwrk[Release] = Rework[Release] * New Error Rate in Rwrk[Release]

Units: Defect/Month

the rate of errors created by rework depend on the error rate and the amount of rework we do.

$$\mathbf{Dfcts\ Left\ in\ RWrk[Release]} = \mathbf{Rework[Release]} * \mathbf{Ave\ Dfct\ in\ Rwrk\ Code[Release]} * (1 - \mathbf{Frac\ Errors\ Found\ in\ Rwrk})$$

Units: Defect/Month

A fraction of errors are not corrected in each iteration of rework, since not all defects are caught by testing at the first place.

$$\mathbf{Dsrđ\ Dev\ Resources[Release]} = \mathbf{Dsrđ\ New\ Dev\ Resources[Release]} + \mathbf{Dsrđ\ Rwrk\ Resources[Release]}$$

Units: Person

Desired development resources for new code and rework.

$$\mathbf{Dsrđ\ New\ Dev\ Resources[Release]} = \mathbf{Max} (\mathbf{Desired\ Development\ Rate[Release]} / \mathbf{Normal\ Productivity[Release]}, 0)$$

Units: Person

Desired new development rate depends on the productivity in normal situations and the rate of development desired.

$$\mathbf{Dsrđ\ Rwrk\ Rate[Release]} = \mathbf{Code\ Waiting\ Rework[Release]} / \mathbf{Dsrđ\ Time\ to\ Do\ Rwrk}$$

Units: module/Month

The desired rework rate depends on the amount of rework in backlog and the how fast we want to do that rework.

$$\mathbf{Dsrđ\ Rwrk\ Resources[Release]} = \mathbf{Max} (0, \mathbf{Dsrđ\ Rwrk\ Rate[Release]} / \mathbf{Normal\ Productivity[Release]})$$

Units: Person

The desired resources for rework depend on the normal productivity of people and the desired rework rate.

$$\mathbf{Dsrđ\ Time\ to\ Do\ Rwrk} = 0.25$$

Units: Month

We want to do our rework in about a week.

$$\mathbf{Eff\ Archtcr\ Complexity\ on\ Productivity} = \mathbf{Max} (1, \mathbf{Accmltd\ Code\ Accepted} / \mathbf{Minimum\ Code\ Size\ With\ an\ Effect}) ^ (\mathbf{Strength\ of\ Complexity\ on\ Productivity\ Effect} * \mathbf{Density\ of\ Problems\ in\ Architecture})$$

Units: Dmnl

The effect of code complexity on productivity not only depends on size of the code base, but also on the quality of the architecture. A power law is assumed for the shape of function.

$$\mathbf{"Eff\ Old\ Code\ \&\ Archtctr\ on\ Error"[Release]} = \mathbf{"Tl\ Eff\ Old\ Code\ \&\ Archtctr\ on\ Error"}$$

(Weighted Error Density in Architecture and Code Base) * SW Eff Old Code and Archtctr on Error + 1 - SW Eff Old Code and Archtctr on Error

Units: Dmnl

Effect of old code and architecture on the quality of development in the current release depends on the overall quality of old code base and architecture.

Eff Pressure on Error Rate[Release] = Tl Eff Pressure on Error Rate (Natural Dev Schedule Pressure[Release]) * SW Eff Pressure on Error + 1 - SW Eff Pressure on Error

Units: Dmnl

Effect of work pressure on error rate is created because under work pressure people take short cuts, skip unit testing, don't study the requirements well enough, don't do code reviews, don't develop as good a design based on given requirements etc. This effect is not only on quality of coding, but also exacerbates the problems with quality that result from poor design and architectural issues.

Eff Pressure on Productivity[Release] = Tl Eff Pressure on Productivity (Natural Dev Schedule Pressure[Release]) * SW Eff Pressure on Productivity + 1 - SW Eff Pressure on Productivity

Units: Dmnl

The effect of schedule pressure on productivity is captured by a table function.

Error per Design Problem in Features = 10

Units: Defect/Problem How many coding defects each design problem can create.

Error Rate New Dev[Release] = Errors from Coding[Release] + Errors from Design and Architecture[Release]

Units: Defect/module

The number of defects found in a module, in average.

Errors from Coding[Release] = Normal Errors from Coding * Eff Pressure on Error Rate[Release] * "Eff Old Code & Archtctr on Error"[Release]

Units: Defect/module

The error rate depends on the inherent quality of the workforce (Normal errors) as well as the work pressure they are working under and the quality of the code and architecture from old releases.

Errors from Design and Architecture[Release] = Ave Prblm in Designed Feature[Release] * Error per Design Problem in Features / Modules for a feature * Eff Pressure on Error Rate[Release]

Units: Defect/module

Errors from design and architecture have a base rate which depend on quality of architecture, but can get worse depending on the work pressure.

Feasible New Dev Rate[Release] = Dev Resources to New Dev[Release] * Productivity[Release]

Units: module/Month

The development rate that is feasible based on availability of resources for development of new code.

Feasible Rwrk Rate[Release] = Dev Resources to Rework[Release] * Productivity[Release]

Units: module/Month

Feasible rework rate from resources depends on the availability of resources as well as their productivity.

Frac Errors Found in Rwrk = 0.7

Units: Dmnl

The defects per module for reworked code.

Frctn Code Cancellation[Release] = TI Eff Schedl Prssr on Cancellation (Current Dev Schedule Pressure[Release]) * Maximum Cancellation Fraction

Units: 1/Month

The cancellation fraction is a function of schedule pressure and can not exceed some maximum level.

"Frctn Des-Dev Overlap" = 0.5

Units: Dmnl

The fraction of overlap between design and development phases is assumed constant, while it really is not. The impact is negligible due to short design phase.

Maximum Cancellation Fraction = 0.5

Units: 1/Month

There is a limit on the cancellation of features feasible.

Min Dev Time = 0.25

Units: Month

The minimum time for development of the remaining code, used when we go beyond schedule.

Minimum Code Size With an Effect = 10000

Units: module

The minimum size of code where effect of complexity is observable.

Modules for a feature = 10

Units: module/feature

Number of modules needed for a new feature.

Natural Dev Schedule Pressure[Release] = ZIDZ (Estmtd Dev Time Left[Release ,Normal] , Development Time Left[Release])

Units: Dmnl

The natural schedule pressure is felt based on what one sees to be the pressure relative to natural productivity.

New Error Rate in Rwrk[Release] = Normal Error Rate in Rwrk * Eff Pressure on Error Rate[Release]

Units: Defect/module

The error rate for creating new errors in the reworked code. It depends on different factors including the work pressure.

Normal Error Rate in Rwrk = 0.5

Units: Defect/module

The normal error rate in rework phase.

Normal Errors from Coding = 0.5

Units: Defect/module

Average number of defects in coding that are created in each module, if the schedule pressure is very low.

Normal Productivity[Release] = 20

Units: module/(Month*Person) [10,30] Normal productivity represents the work rate of an average worker, if s/he follows good processes and does not take any shortcuts.

Productivity[Release] = Normal Productivity[Release] * Eff Pressure on Productivity[Release] * (1 / Eff Archtcr Complexity on Productivity * SW Eff Archtcr Complexity on Productivity + 1 - SW Eff Archtcr Complexity on Productivity)

Units: module/(Month*Person)

The productivity depends on the work pressure and code complexity effects.

Rework[Release] = Feasible Rwrk Rate[Release]

Units: module/Month

Rework rate depends on feasible rate given by the current resource availability.

Sanctioned Code Complete[Release] = INTEG(Dev Reschedule[Release] , 0)

Units: Month

This is the time people think is OK to have the code complete, based on the changing pressures in the field. It is not clear if the code complete is so well set.

Strength of Complexity on Productivity Effect = 1

Units: feature/Problem

The exponent for the power law governing the effect of complexity on productivity

SW Eff Archtcr Complexity on Productivity = 0

Units: Dmnl

Put 1 to get the effect of code complexity and architecture quality on productivity, put 0 to avoid that.

SW Eff Old Code and Archtctr on Error = 1

Units: Dmnl

Put 1 to have the effect of old code and architecture on error rate on, put zero to remove that effect.

SW Eff Pressure on Error = 1

Units: Dmnl

Put 1 to get the feedback from schedule pressure on error rate. Put 0 to get no such feedback.

SW Eff Pressure on Productivity = 1

Units: Dmnl

Put 1 to have the effect of work pressure on productivity on, put 0 to switch that effect off.

Test Acceptance[Release] = Code Passing Test[Release] - Test Rejection[Release]

Units: module/Month

The code accepted in testing is what is not rejected, which in turn depends on the coverage as well as total test done.

Test Rejection[Release] = Totl Code Tstd[Release] * Tst Rjctn Frac[Release]

Units: module/Month

Testing rejects in a fraction of cases, and therefore creates proportional amount of rework.

"TI Eff Old Code & Archtctr on Error" ([(0,0)-(1,3)],(0,1),(0.16208,1.17105), (0.333333,1.68421),(0.480122,2.31579),(0.663609,2.75),(0.844037,2.92105) ,(1,3))

Units: Dmnl

The shape of this table function might be not S-shape since the worse the old architecture is, the more problematic new coding will be, yet there should be some saturation level, hence the current formulation.\!\\!

TI Eff Pressure on Error Rate ([(0,0)-(4,4)],(0,1),(1,1),(1.27217,1.22807), (1.48012,1.5614),(1.71254,2),(2.01835,2.36842),(2.42202,2.72368) ,(3,3),(10,4))

Units: Dmnl

Table for effect of time pressure on error rate.

TI Eff Pressure on Productivity ([(0,0)-(3,2)],(0,0.8),(0.623853,0.885965), (1,1),(1.27523,1.16667),(1.54128,1.38596),(1.85321,1.54386),(2.37615,1.65789), (3,1.7),(10,2))

Units: Dmnl

Table for the effect of time pressure on productivity.

TI Eff Schdul Prssr on Cancellation ([(0,0)-(10,1)],(0,0),(1,0),(2,0),(10,0))

Units: Dmnl

This table represents the reaction of the development group by cancelling features to the schedule pressure they perceive.

$$\text{Total Development[Release]} = \text{Development[Release]} + \text{Rework[Release]}$$

Units: module/Month

Total development consists of new development and the rework for the current release.

$$\text{Totl Code to Tst[Release]} = \text{Code Developed[Release]} + \text{Code to Develop[Release]} + \text{Code Waiting Rework[Release]}$$

Units: module

The total code that needs testing include those pieces developed, those under rework, and the parts that need to be developed.

$$\text{Totl Code Tstd[Release]} = \text{Code Covered by Test} * \text{Test Rate[Release]}$$

Units: module/Month

the total amount of code that is really tested, depends on the number of tests and the coverage of each test.

$$\text{Tst Effectiveness} = 0.9$$

Units: Dmnl

The effectiveness of tests in finding bugs, if they are in the area that is tested.

$$\text{Tst Rjctn Frac[Release]} = 1 - \text{EXP} (- \text{Tst Effectiveness} * \text{Ave Dfct in Tstd Code[Release]})$$

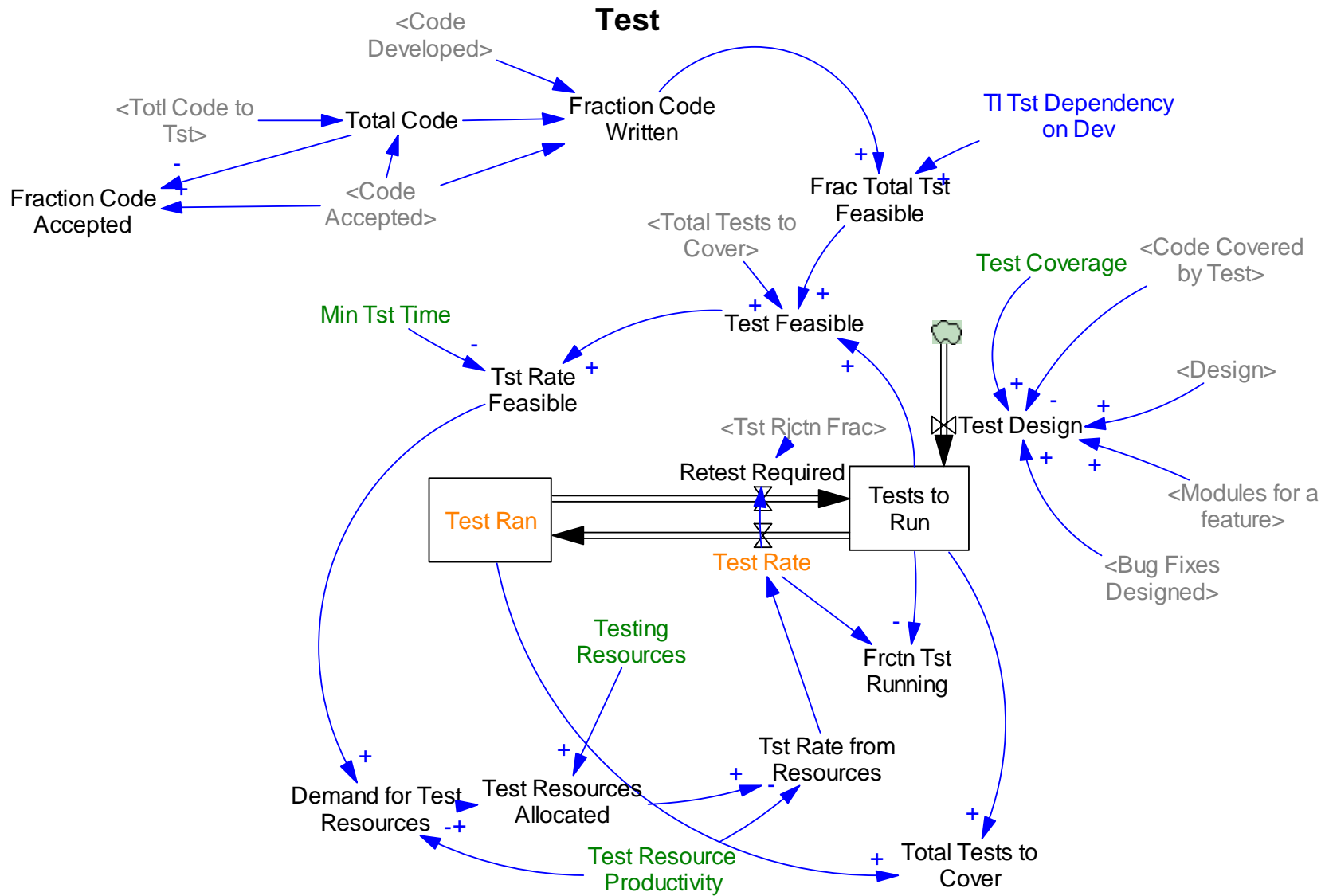
Units: Dmnl

The fraction of tests that reject are calculated based on the assumption that defects are randomly distributed in the whole code base. Therefore the number of defects per test is a Poisson distribution with mean of 'Ave Dfct in Tstd Code'. Assuming an independent chance of 'Tst Effectiveness' for finding each of those errors, we can find the chance of being rejected by finding the summation of that chance over all number of errors (0 to infinity).

$$\text{Tstd Code Accptd[Release]} = \text{Test Rate[Release]} * \text{Code Covered by Test} * (1 - \text{Tst Rjctn Frac[Release]})$$

Units: module/Month

The portion of tested code that gets accepted, depends on test coverage and the fraction of acceptance.



Test

Demand for Test Resources[Release] = Tst Rate Feasible[Release] / Test Resource Productivity

Units: Person

The total demand for test resources across ongoing releases.

Frac Total Tst Feasible[Release] = Tl Tst Dependency on Dev[Release] (Fraction Code Written[Release])

Units: Dmnl

This is the fraction of total tests that are ready to be executed. We can't run tests without having them ready, because of interdependencies etc. Blocking issues are included in this table function.

Fraction Code Accepted[Release] = ZIDZ (Code Accepted[Release] , Total Code[Release])

Units: Dmnl

The fraction of the total code that is done and tested.

Fraction Code Written[Release] = ZIDZ (Code Accepted[Release] + Code Developed[Release] , Total Code[Release])

Units: Dmnl

Fraction of code that is written and either tested or ready to test. This determines the fraction of tests one can do. Note that the inputs to this do not include any delays, which assumes that the delays are relatively small for test to know where the code is.

Frctn Tst Running[Release] = ZIDZ (Test Rate[Release] , Tests to Run[Release])

Units: 1/Month

The fraction of total tests that is being run at this time

Min Tst Time = 0.1

Units: Month

The minimum time needed to run a test.

Retest Required[Release] = Test Rate[Release] * Tst Rjctn Frac[Release]

Units: Test/Month

The tests that are rejected need to be redone, after the code is revisited.

Test Coverage = 0.5

Units: Dmnl

How much of the code the test set to be designed covers

Test Design[Release] = (Design[Release] * Modules for a feature + Bug Fixes Designed[Release]) / Code Covered by Test * Test Coverage

Units: Test/Month

The tests are assumed to be designed as we design the features or bug fixes. So test design depends on how fast we develop the requirements, what is our test coverage, how big is each requirement, and how much code each test covers.

Test Feasible[Release] = Max (0, Frac Total Tst Feasible[Release] * Total Tests to Cover[Release] - (Total Tests to Cover[Release] - Tests to Run[Release]))

Units: Test

The feasible tests are those that are not already done and are feasible based on the precedence relationships.

Test Ran[Release] = INTEG(Test Rate[Release] - Retest Required[Release] , 0)

Units: Test

The number of tests that are already done and have passed. In coflow with code, it parallels the code accepted stock.

Test Rate[Release] = Tst Rate from Resources[Release]

Units: Test/Month

The test rate depends on both availability of resources that are requested based on feasibility of tests considering the current development level

Test Resource Productivity = 50

Units: Test/(Month*Person)

The productivity of test personnel for doing tests.

Test Resources Allocated[Release] = Min (Demand for Test Resources[Release] , ZIDZ (Demand for Test Resources[Release] * Testing Resources , SUM (Demand for Test Resources[Release!])))

Units: Person

The test resources are given to the release based on availability of those resources and proportional to requests.

Testing Resources = 20

Units: Person

The total number of testing resources available. Assumed to be fixed here.

Tests to Run[Release] = INTEG(Retest Required[Release] + Test Design[Release] - Test Rate[Release] , 0)

Units: Test

Note that in a coflow of tests with code, this stock represents all the code that is not yet administered, i.e. code to develop, developed, or waiting rework.

TI Tst Dependency on Dev[Release] ([(0,0)-(1,1)],(0,0),(0.9,0),(0.95,0.1), (1,1),(1.1,1))

Units: Dmnl

The shape of the curve suggests what fraction of our tests we can do based on the current level of development.

$$\mathbf{Total\ Code[Release]} = \text{Code Accepted[Release]} + \text{Totl Code to Tst[Release]}$$

Units: module

The total code

$$\mathbf{Total\ Tests\ to\ Cover[Release]} = \text{Test Ran[Release]} + \text{Tests to Run[Release]}$$

Units: Test

Total number of tests that are designed and can be run.

$$\mathbf{Tst\ Rate\ Feasible[Release]} = \text{Test Feasible[Release]} / \text{Min Tst Time}$$

Units: Test/Month

The maximum feasible testing rate depend on what amount of tests we can potentially administer and how long each test takes an minimum.

$$\mathbf{Tst\ Rate\ from\ Resources[Release]} = \text{Test Resources Allocated[Release]} * \text{Test}$$

Resource Productivity

Units: Test/Month

The resources for testing can put an upper bound on the amount of testing we can do.

Defects in the Field

Ave Def with Patch[Release] = ZIDZ (Defects with Patch[Release] , Sites Installed[Release])

Units: Defect

The average number of defects in sites for which we have already developed a patch

Ave Known Def in Site[Release] = ZIDZ (Known Defects in Field[Release] , Sites Installed[Release])

Units: Defect

The number of defects known in a typical site.

Ave Unknown Def in Site[Release] = ZIDZ (Unknown Defects in Field[Release] , Sites Installed[Release])

Units: Defect

The average number of unknown defects that is in use in a typical site

Average Defects Patched[Release] = ZIDZ (Defects Patched[Release] , Sites Installed[Release])

Units: Defect

The average number of defects patched in a typical site.

Average Outstanding Defects = Total Ave Defect - SUM (Average Defects Patched[Release!])

Units: Defect

This is the number of defects outstanding in the sites, an overall measure of quality

Average Time in the Field = 30

Units: Month

The product remains in the field for about 50 month

Calls from Defects With Patch[Release] = Defects with Patch[Release] / Time to Show up for Def with Patch

Units: Defect*Site/Month

Number of calls coming from defects for which a patch exist

Calls from Known Defects[Release] = Known Defects in Field[Release] / Time to Show Up for Defects

Units: Defect*Site/Month

The number of defects that show up again as customer calls, even though we know about them.

Calls from Unknown Defects[Release] = Unknown Defects in Field[Release] / Time to Show Up for Defects

Units: Defect*Site/Month

The number of MRs created by real, unknown problems in the customer site.

Defect Covering by Patch[Release] = Patch Development[Release] * Defect per Patch

Units: Defect/Month

The total number of defects that are covered by these patches

Defect not Patching[Release] = Defect Covering by Patch[Release] * Sites Installed[Release] * (1 - Fraction Sites Installing)

Units: Defect*Site/Month

The number of defects for which we have developed patches, yet people have not installed those patches.

Defect Patching[Release] = Defect Covering by Patch[Release] * Sites Installed[Release] * Fraction Sites Installing

Units: Defect*Site/Month

The number of defects that are patched in different installation of the software in the field.

Defect Show up[Release] = Calls from Unknown Defects[Release] * Sites Installed[Release] / Sites Reporting One Defect[Release]

Units: Defect*Site/Month

Every defect that is found becomes known and therefore is known for all the sites installed.

Defects Patched[Release] = INTEG(Defect Patching[Release] + Installing Known Patches[Release] - Patches Retired[Release] + Patched Defects Sold[Release] , 0)

Units: Defect*Site

I assume there is no patches at the beginning of simulation, therefore there is no defect that is covered by patches

Defects Retired[Release] = Products Retired[Release] * Ave Unknown Def in Site[Release]

Units: Defect*Site/Month

As products are retired, some of the defects that have remained unknown also retire with the solution.

Defects with Patch[Release] = INTEG(Defect not Patching[Release] - Defects with Patch Retired[Release] - Installing Known Patches[Release] + Unpatched Defects Sold with Patch[Release] , 0)

Units: Defect*Site

I assume there is no patches at the beginning of simulation, therefore there is no defect that is covered by patches

Defects with Patch Retired[Release] = Products Retired[Release] * Ave Def with Patch[Release]

Units: Defect*Site/Month

Defects with developed patches are retired as installations are retired

Fraction Defect Unknown[Release] = $XIDZ$ (Unknown Defects in Field[Release] , Total Defects in Field[Release] , 1)

Units: Dmnl

Fraction of defects which are unknown.

Fraction of Defects Important = 0.15

Units: Dmnl

The fraction of total defects in the code shipped that can lead to customer calls and quality issues.

Fraction of sites Installing All patches for a Call = 0.1

Units: 1/Defect

Each defect that escalates results in the site installing the available patches or not. This number indicates what fraction of sites (or patches) are installed following each call.

Fraction Patches Installed At Sales[Release] = 0.8

Units: Dmnl

What fraction of already known problems patched, are fixed at the installation time.

Fraction Sites Installing = 0.5

Units: Dmnl

What fraction of sites we can persuade to install the patches at the time they are out.

Installing Known Patches[Release] = Total Customer Calls from Defects[Release] * Fraction of sites Installing All patches for a Call * Ave Def with Patch[Release]

Units: Defect*Site/Month

Any customer call that escalates results in installing the available patches on the system, therefore removing the possible issues created by those patches

Known Defects in Field[Release] = $INTEG$ (Defect Show up[Release] - Defect not Patching[Release] - Defect Patching[Release] - Known Defects Retired[Release] + Known Defects Sold[Release] , Defect Show up[Release] * Time to Show Up for Defects)

Units: Defect*Site

The total number of known defects, coming from unknown defects discovered or known defects sold.

Known Defects Retired[Release] = Products Retired[Release] * Ave Known Def in Site[Release]

Units: Defect*Site/Month

Known defects are retired as we retire the systems installed.

Known Defects Sold[Release] = Total Defects Introduced[Release] * XIDZ (Known Defects in Field[Release] , Total Defects in Field[Release] , 0)

Units: Defect*Site/Month

The total number of known defects introduced since we know some defects exist when we are introducing the product.

Patched Defects Sold[Release] = Total Defects Introduced[Release] * ZIDZ (Defects Patched[Release] + Defects with Patch[Release] , Total Defects in Field[Release]) * Fraction Patches Installed At Sales[Release]

Units: Defect*Site/Month

Of total defects for which we have already made a patch, a fraction will be sold with the patches installed.

Patches Retired[Release] = Products Retired[Release] * Average Defects Patched[Release]

Units: Defect*Site/Month

Defects we have fixed through patches also retire with retired installments.

Products Retired[Release] = Sites Installed[Release] / Average Time in the Field

Units: Site/Month

The rate of retiring the systems depends on the number of systems we have installed and how long they last in the field.

Sites Installed[Release] = INTEG(Products Sold[Release] - Products Retired[Release] , Products Sold[Release] * Average Time in the Field)

Units: Site/Month

The total number of sites with a version of the product installed in their site.

Sites Reporting One Defect[Release] = 1

Units: Site/Month

The number of sites that report a single bug

Time to Show up for Def with Patch = 10

Units: Month

The defects for which patches are developed but not installed usually get longer to trigger any problem.

Time to Show Up for Defects = 50

Units: Month

The time it takes for an average defect to show up

Total Ave Defect = SUM (Ave Def with Patch[Release!] + Ave Known Def in Site[Release!] + Ave Unknown Def in Site[Release!] + Average Defects Patched[Release!])

Units: Defect

Total number of average defects, distributed between different stocks.

Total Defects in Field[Release] = Defects Patched[Release] + Defects with Patch[Release] + Known Defects in Field[Release] + Unknown Defects in Field[Release]

Units: Defect*Site

The total number of patched or unpatched defects in the field

Total Defects Introduced[Release] = Products Sold[Release] * (Defects in Accepted Code[Release] + Defects in Code Developed[Release] + Defects in MRs[Release]) * Fraction of Defects Important

Units: Defect*Site/Month

The observable defects introduced depend on the rate of sales, the average number of defects in an installation, and the fraction of those defects that can create observable issues for the customer.

Unknown Defects in Field[Release] = INTEG(Unknown Defects Introduced[Release] - Defect Show up[Release] - Defects Retired[Release] , 0)

Units: Defect*Site

The total number of important defects over all different sites.

Unknown Defects Introduced[Release] = Total Defects Introduced[Release] * Fraction Defect Unknown[Release]

Units: Defect*Site/Month

Unknown defects get introduced since several of the defects are unknown yet!

Unpatched Defects Sold with Patch[Release] = Total Defects Introduced[Release] * ZIDZ ((Defects with Patch[Release] + Defects Patched[Release]) * (1 - Fraction Patches Installed At Sales[Release]) , Total Defects in Field[Release])

Units: Defect*Site/Month

A fraction of total patches are installed at the time of sales to new sites.

Patches and Current Engineering

Accumulated Customer Calls = INTEG(Inc Customer Calls , 0)

Units: Defect*Site

The accumulated number of customer calls from all the releases until this point in time.

Accumulated Total Work = INTEG(Development and CE Rate , 0)

Units: module

The total accumulated work done on the product, both on development and patching.

Allctd Ttl Dev Resource = Min (Ttl Des Resource to CR , XIDZ (Ttl Des Resource to CR * Total Dev Resources , Total Desired Resources , Total Dev Resources))

Units: Person

We allocate the resources between current engineering and new release proportionately

Allctd Ttl CE Resource = Min (Ttl Des Resource to CE , ZIDZ (Ttl Des Resource to CE * Total Dev Resources , Total Desired Resources))

Units: Person

Resources are allocated proportional to request, unless we have more resource than needed.

CR Dev Resources Left After Allocation[R1] = Max (Allctd Ttl Dev Resource - Dsrdev Resources[R1] , 0)

CR Dev Resources Left After Allocation[Later Releases] = Max (0, VECTOR ELM MAP (CR Dev Resources Left After Allocation[R1] , Later Releases - 2) - Dsrdev Resources[Later Releases])

Units: Person

The amount of resources left for new development resources given to different future releases, after allocating the resources to all the releases before that. This is based on the allocation scheme where we give all the resources requested by the most mature release, before going to the next.

Desired Patch Rate[Release] = Patch to Develop[Release] / Time to Patch

Units: Patch/Month

Desired speed of patching.

Desired Resources to Current Engineering[Release] = Desired Patch Rate[Release] / Productivity in Writing Patch[Release]

Units: Person

Desired resources for current engineering.

Desired Total Dev Resources = 50

Units: Person

This is the total maximum resources desired to be given to this product for development and current engineering. In fact it should be tied to the amount of work expected for the product, not a stand-alone concept.

Development and CE Rate = SUM (Total Development[Release!] + Patch Development[Release!] * Modules Equivalent to Patch)

Units: module/Month

The rate of development work done on the product in total (over all releases)

Development Resource to Current Engineering[Release] = Desired Resources to Current Engineering[

Release] * Frac CE Allctd

Units: Person

The development resources allocated to developing patches across different releases are proportional to their requests.

Development Resources to Current Release[Later Releases] = Dsrdev Resources[Later Releases] * Frac CR Allctd * (1 - SW Higher Priority to Current) + SW Higher Priority to Current * if then else (CR Dev Resources Left After Allocation[Later Releases] = 0, VECTOR ELM MAP (CR Dev Resources Left After Allocation[R1] , Later Releases - 2) , Dsrdev Resources[Later Releases])

Development Resources to Current Release[R1] = Dsrdev Resources[R1] * Frac CR Allctd * (1 - SW Higher Priority to Current) + SW Higher Priority to Current * if then else (CR Dev Resources Left After Allocation[R1] = 0, Allctd Ttl Dev Resource , Dsrdev Resources[R1])

Units: Person

We allocate the resources to each release proportional to their requests for resources. Alternatively one can allocate resources to the sooner release in the expense of the older one. Both allocations are possible here and one can choose between them using the switch.

Eff Profitability on Resources Allocated = (Tl Eff Profit on Resources (Perceived Profitability) * Min (Time Since Release Ready[R1] / "Honey-Moon Period" , 1) + 1 - Min (Time Since Release Ready[R1] / "Honey-Moon Period" , 1)) * SW Eff Profit on Resources + 1 - SW Eff Profit on Resources

Units: Dmnl

We gradually strengthen the tie between profitability and resources allocated, as the project gets more mature (passes through honeymoon period).

Frac CE Allctd = ZIDZ (Alloctd Ttl CE Resource , Ttl Des Resource to CE) * (1 - SW Exogenous CE Resources) + SW Exogenous CE Resources

Units: Dmnl

Fraction of CE resource request that is allocated

Frac CR Allctd = XIDZ (Allctd Ttl Dev Resource , Ttl Des Resource to CR , 1)

Units: Dmnl

The fraction of total resources for current development that is allocated.

Fraction of Calls Indicating New Defects[Release] = $ZIDZ$ (Calls from Unknown Defects[Release] , Total Customer Calls from Defects[Release])

Units: Dmnl

Fraction of calls that indicate a new defect.

Fraction of Desired CR Allocated[Release] = $XIDZ$ (Development Resources to Current Release[Release] , Dsrdev Resources[Release] , 1)

Units: Dmnl

What fraction of desired resources for current development of each release is allocated to them.

"Honey-Moon Period" = 20

Units: Month

The period during which the effect of profitability on resources is not strong

Inc Customer Calls = SUM (Total Customer Calls from Defects[Release!])

Units: Defect*Site/Month

Increase in the number of customer calls.

Patch Developed[Release] = (Ave Def with Patch[Release] + Average Defects Patched[Release]) / Defect per Patch

Units: Patch

The total number of patches developed so far for different releases of the product

Patch Development[Release] = Development Resource to Current Engineering[Release] * Productivity in Writing Patch[Release]

Units: Patch/Month

The rate of developing patches depends both on the productivity of developers in writing patches and on quantity of developers devoted to this task.

Patch to Develop[Release] = Ave Known Def in Site[Release] / Defect per Patch

Units: Patch

The stock of patches pending development

Perceived Profitability = $INTEG$ ((Profitability - Perceived Profitability) / Time to Perceive Profitability , 0)

Units: Dmnl

Perceived profitability fraction.

Productivity in Writing Patch[Release] = Normal Productivity[Release] / Modules Equivalent to Patch

Units: Patch/(Month*Person)

The number of patches an individual can write in a month. The productivity should be connected to what the productivity effects are for development, however, effect of work

pressure is not so high since patches are written in a fairly high pressure environment anyway (customers are waiting) and effect of code complexity is not so big since people try to do the patches with the least interaction. It is red to make sure that other possible feedbacks are included in it.

SW Eff Profit on Resources = 0

Units: Dmnl

Put 1 to have the effect of profitability on resources allocated, put 0 to cut this feedback.

SW Exogenous CE Resources = 0

Units: Dmnl

Put 0 to get normal allocation to CE, put 1 to get exogenous allocation to CE, where all the resources needed are given to CE, without interfering with the resources for current development'

SW Higher Priority to Current = 1

Units: Dmnl

Put 1 so that allocation of resources for development of different releases gives categorically higher priority to more eminent ones, put 0 for proportional distribution of resources between different releases.

Time Since Release Ready[Release] = INTEG("Release Ready?"[Release] , 0)

Units: Month

The number of month since each release is ready.

Time to Adjust Resources = 10

Units: Month

Time to adjust the resources to what is dictated by profitability concerns.

Time to Patch = 5

Units: Month

The desired time to write the patches for the outstanding defects

Time to Perceive Profitability = 10

Units: Month

Time constant for perceiving profitability.

TI Eff Profit on Resources ([(-1,0)-(1,1.3)],(-1,0.5),(-0.425076,0.587281) ,(-0.125382,0.695614),(0,0.8),(0.1,1),(0.174312,1.06623),(0.29052,1.10614) ,(0.535168,1.15746),(1,1.2))

Units: Dmnl

Table for effect of profitability on resources given to the project.

Total Allocated Resources = Allotted Ttl Dev Resource + Allocated Ttl CE Resource

Units: Person

Total resources allocated for development and current engineering.

Total Customer Calls from Defects[Release] = Calls from Defects With Patch[Release] + Calls from Known Defects[Release] + Calls from Unknown Defects[Release]

Units: Defect*Site/Month

Total customer calls coming from defects in the product

Total Desired Resources = Ttl Des Resource to CR + Ttl Des Resource to CE

Units: Person

The total development and CE resources requested

Total Dev Resources = INTEG((Desired Total Dev Resources * Eff Profitability on Resources Allocated - Total Dev Resources) / Time to Adjust Resources , Desired Total Dev Resources)

Units: Person

The resources allocated depend to the profitability of the product to some extent.

Ttl Des Resource to CR = SUM (Dsrld Dev Resources[Release!])

Units: Person

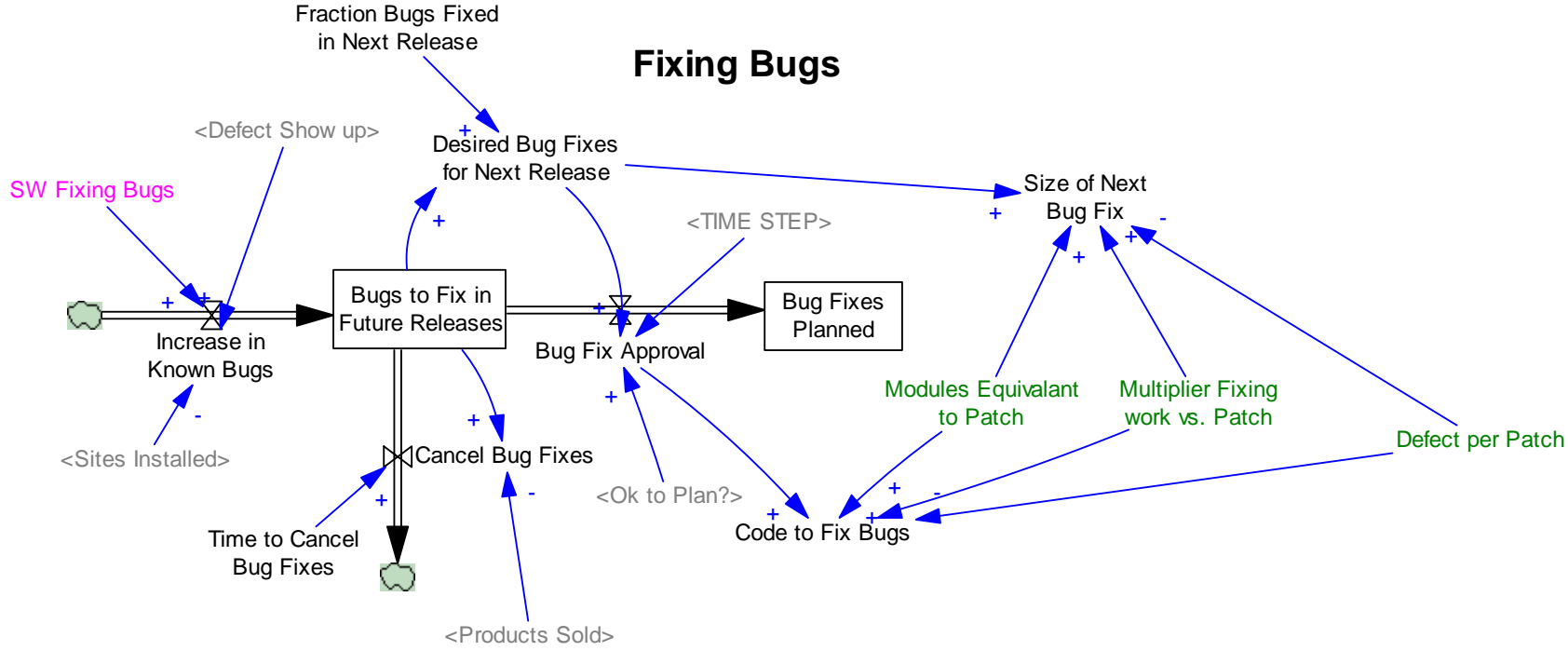
Total people requested for the development tasks at hand for the currently under development releases

Ttl Des Resource to CE = SUM (Desired Resources to Current Engineering[Release!]) * (1 - SW Exogenous CE Resources)

Units: Person

Total people requested for the development tasks at hand for the current engineering of different releases.

Fixing Bugs



Fixing Bugs

Bug Fix Approval[Release] = SUM (Desired Bug Fixes for Next Release[Release!]) * "Ok to Plan?"[Release] / TIME STEP

Units: Defect/Month

The bug fixes planned are assumed to be decided upon at the OK to plan. We assume that we take bug fixes proportionately from all the different releases.

Bug Fixes Planned[Release] = INTEG(Bug Fix Approval[Release] , 0)

Units: Defect

Number of bug fixes planned in each release.

Bugs to Fix in Future Releases[Release] = INTEG(Increase in Known Bugs[Release] - ZIDZ (Bugs to Fix in Future Releases[Release] , SUM (Bugs to Fix in Future Releases[Release!])) * SUM (Bug Fix Approval[Release!]) - Cancel Bug Fixes[Release] , 0)

Units: Defect

This is parallel to stock of open MRs from all different releases. The current formulation does not reflect how quality of future releases for customers is increased by doing bug fixes in this release. That is usually one of the important motivations for having bug fixes in a release.

Cancel Bug Fixes[Release] = Bugs to Fix in Future Releases[Release] / Time to Cancel Bug Fixes * (1 - ZIDZ (Products Sold[Release] , SUM (Products Sold[Release!])))

Units: Defect/Month

We cancel bug fixes based on the importance of the product in the market. If a product is selling well, we will put more emphasis on fixing its bugs, than when a release is not selling so well (in which case we start draining the bug fixes for that release).

Code to Fix Bugs[Release] = Bug Fix Approval[Release] / Defect per Patch * Modules Equivalent to Patch * "Multiplier Fixing work vs. Patch"

Units: module/Month

The number of modules of code needed to be written as a result of bug fixes planned is determined by looking at the number of patches and the adjusting the scope to account for the fact that bug fixes are more comprehensive and time consuming.

Defect per Patch = 4

Units: Defect/Patch

The number of defects that are taken care of in one typical patch

Desired Bug Fixes for Next Release[Release] = Bugs to Fix in Future Releases[Release] * Fraction Bugs Fixed in Next Release

Units: Defect

The number of MRs planned to be fixed in the upcoming release.

Fraction Bugs Fixed in Next Release = 0.3

Units: Dmnl

Fraction of bugs normally planned to be fixed in the next release.

Increase in Known Bugs[Release] = ZIDZ (Defect Show up[Release] , Sites Installed[Release]) * SW Fixing Bugs

Units: Defect/Month

The increase in the number of bugs we know about in the field depends on the rate of bug reports from the field across all releases.

Modules Equivalent to Patch = 20

Units: module/Patch A measure of how much work a patch entails, when compared to a typical module.

"Multiplier Fixing work vs. Patch" = 2

Units: Dmnl

Fixing the errors when done as part of a new release and included in the whole code is this much more resource consuming than fixing them as patches.

Size of Next Bug Fix = SUM (Desired Bug Fixes for Next Release[Release!]) / Defect per Patch * Modules Equivalent to Patch * "Multiplier Fixing work vs. Patch"

Units: module

Size of the next bug fix in the upcoming planning of a release is determined by desired number of bug fixes.

SW Fixing Bugs = 1

Units: Dmnl

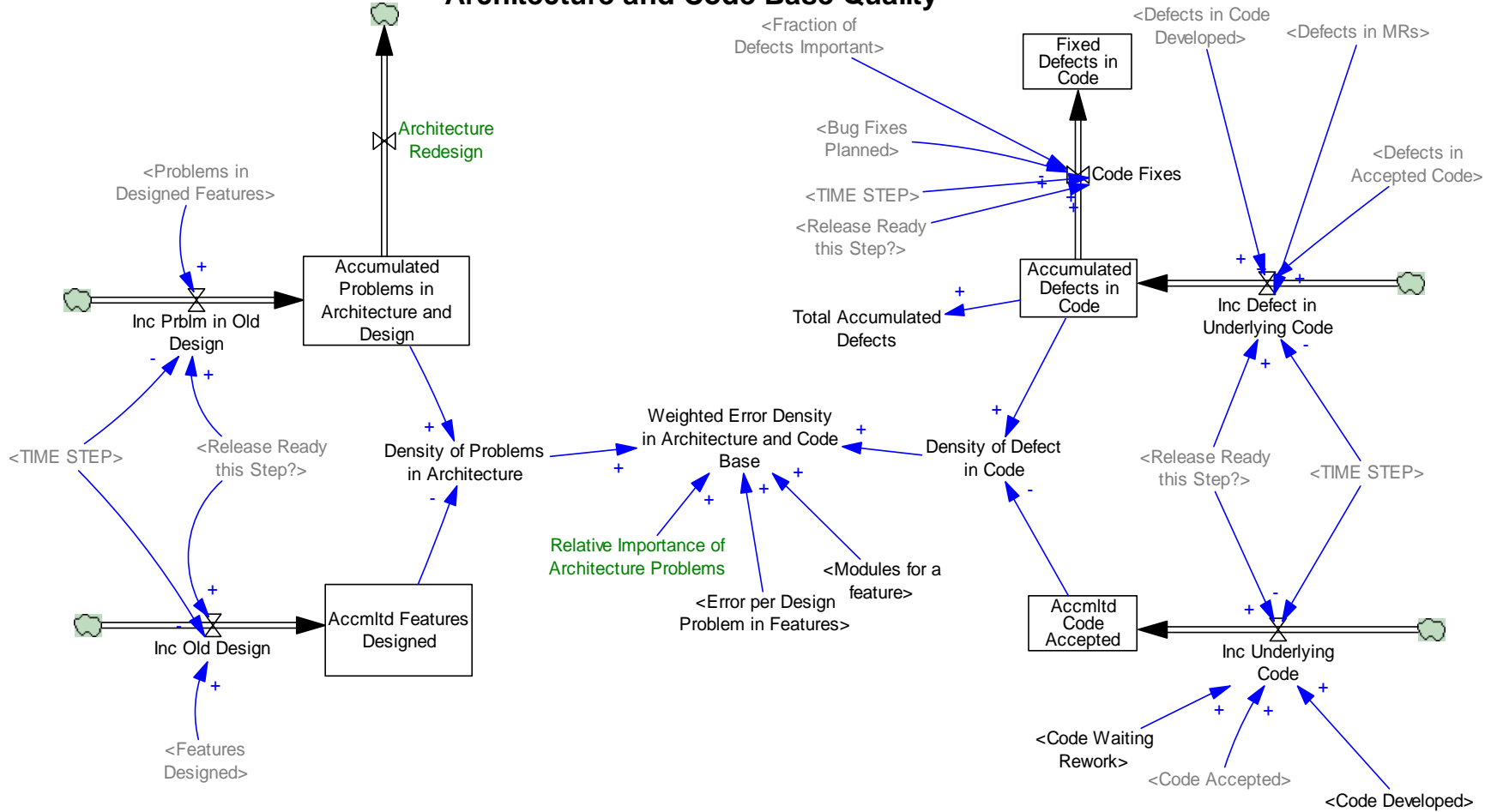
Put 1 to have bugs from old release become part of new release planning. Put 0 so that such effect does not exist, i.e. the old bugs are completely ignored.

Time to Cancel Bug Fixes = 10

Units: Month

The time frame for draining the stock of bugs to fix.

Architecture and Code Base Quality



Architecture and Code Base Quality

Accmltd Code Accepted = INTEG(Inc Underlying Code , 0)
Units: module
Accumulated code accepted.

Accmltd Features Designed = INTEG(Inc Old Design , 0)
Units: feature
Accumulated number of features designed.

Accumulated Defects in Code = INTEG(Inc Defect in Underlying Code - Code Fixes , 0)
Units: Defect
The total number of defected codes accumulated across all releases

Accumulated Problems in Architecture and Design = INTEG(Inc Prblm in Old Design - Architecture Redesign , 0)
Units: Problem
The accumulated problems in the architecture and design that influence the quality of future releases.

Architecture Redesign = 0
Units: Problem/Month
The refactoring of architecture can reduce the errors in it.

Code Fixes = SUM (Bug Fixes Planned[Release!] / TIME STEP * "Release Ready this Step?"[Release!]) / Fraction of Defects Important
Units: Defect/Month
The rate of fixing the code base as a result of incorporating MR fixes in the new release. Note that that no matter what fraction of defects we call important, we will transfer same amount from this stock.

Density of Defect in Code = ZIDZ (Accumulated Defects in Code , Accmltd Code Accepted)
Units: Defect/module
The density of defects in the underlying legacy code.

Density of Problems in Architecture = ZIDZ (Accumulated Problems in Architecture and Design , Accmltd Features Designed)
Units: Problem/feature
The density of problems in the features already designed and used as building block of future code.

Fixed Defects in Code = INTEG(Code Fixes , 0)

Units: Defect
accumulated number of fixed (former) defects in the code.

Inc Defect in Underlying Code = $\text{SUM} ((\text{Defects in Accepted Code}[\text{Release!}] + \text{Defects in Code Developed}[\text{Release!}] + \text{Defects in MRs}[\text{Release!}]) * \text{"Release Ready this Step?"}[\text{Release!}]) / \text{TIME STEP}$

Units: Defect/Month
Defects in the code that is released become influential in the quality of upcoming releases, to the extent that they hinder good development.

Inc Old Design = $\text{SUM} (\text{Features Designed}[\text{Release!}] / \text{TIME STEP} * \text{"Release Ready this Step?"}[\text{Release!}])$

Units: feature/Month
The addition of legacy design

Inc Prblm in Old Design = $\text{SUM} (\text{Problems in Designed Features}[\text{Release!}] / \text{TIME STEP} * \text{"Release Ready this Step?"}[\text{Release!}])$

Units: Problem/Month
Rate of increase in the problems of the underlying design.

Inc Underlying Code = $\text{SUM} ((\text{Code Accepted}[\text{Release!}] + \text{Code Developed}[\text{Release!}] + \text{Code Waiting Rework}[\text{Release!}]) * \text{"Release Ready this Step?"}[\text{Release!}]) / \text{TIME STEP}$

Units: module/Month
Rate of increase in the underlying code

Relative Importance of Architecture Problems = 2

Units: Dmnl
This parameter factors in the difference between one problem in a feature architecture, vs. one defect in a module of code written. The higher the parameter, the more important the weight of architecture is in this balance.

Total Accumulated Defects = Accumulated Defects in Code

Units: Defect
Total number of accumulated defects in the underlying code.

Weighted Error Density in Architecture and Code Base = $(\text{Density of Defect in Code} + \text{Density of Problems in Architecture} * \text{Error per Design Problem in Features} / \text{Modules for a feature} * \text{Relative Importance of Architecture Problems}) / (1 + \text{Relative Importance of Architecture Problems})$

Units: Defect/module
The weighted error density in architecture and code base shows the density of errors in old architecture and code base as it influences the quality of development in new releases.

Scheduling

Accumulated Change in Release Date[Release] = INTEG(Chang in Release Date[Release] , 0)

Units: Month

The amount of change in the release date accumulated through time for different releases.

Allocated Design Finish Date[Release] = Time + Allocated Design Time Left[Release]

Units: Month

Allocated design finish date, used to determine the development finish date.

Allocated Dev Finish Date[Release] = Time + (Allocated Design Finish Date[Release] - Time) * (1 - "Frctn Des-Dev Overlap") + Allocated Dev Time Left[Release]

Units: Month

The development finish date allocated.

Allocated Dev Time Left[Release] = Estmtd Dev Time Left[Release,Current] * Prcnt Time Allocated[Release]

Units: Month

Allocated development time left.

Allocated Release Date[Release] = Allocated Dev Finish Date[Release] + Allocated Test Time Left[Release] * (1 - "Frctn Test-Dev Overlap")

Units: Month

Allocated release date, based on development, design, and testing phase conditions.

Allocated Test Time Left[Release] = Estmtd Test Time Left[Release] * Prcnt Time Allocated[Release]

Units: Month

Allocated testing time left.

Chang in Release Date[Release] = (Allocated Release Date[Release] - Sanctioned Release Date[Release]) / Time to Replan * if then else (Sanctioned Release Date[Release] = 0, 0, 1) * (1 - "Release Ready?"[Release])

Units: Dmnl

This is the change rate in the release date, which accumulates to show the delay in one release.

Chng Prcvd Estmtd Code[Release] = if then else ("Release Started?"[Release] = 1 :AND: Perceived Estimated Code to Write[Release] = 0, Estmtd Code to Write[Release] / TIME STEP , (Estmtd Code to Write[Release] - Perceived Estimated Code to Write[Release]) / Time to Perceive Current Status)

Units: module/Month

This equation ensures that the perceived estimated release date starts at estimated level at the beginning but then adjusts to it with some delay.

$$\text{Chng Status[Release]} = (1 - \text{"Release Ready?"[Release]}) * \text{"Release Ready this Step?"[Release]} / \text{TIME STEP}$$

Units: 1/Month

The rate to change the status of a release into ready.

$$\text{Estimated Test Left[Release]} = \text{Estmtd Tests to Be Designed[Release]} + \text{Tests to Run[Release]}$$

Units: Test

The number of tests to run.

$$\text{Estmtd Code to Write[Release]} = \text{Code to Develop[Release]} + \text{Features to Design[Release]} * \text{Modules for a feature}$$

Units: module

A rough estimate of the code waiting to be written comes from currently available code and the part to be designed based on the current specifications.

$$\text{Estmtd Design Time Left[Release]} = \text{ZIDZ} (\text{Features to Design[Release]} , \text{Features Designed[Release]} + \text{Features to Design[Release]}) * \text{Normal Design Time}$$

Units: Month

Estimated design time left based on how far in the design process we are.

$$\text{Estmtd Dev Time Left[Release,Productivity Base]} = \text{ZIDZ} (\text{Expctd Code to Write After Being First[Release,Productivity Base]} , \text{Expctd Resource Available to First Release New Dev} * \text{Prcvd Productivity[Release,Productivity Base]})$$

Units: Month

The development time left depends on the work left, the expected resources available and productivity of those resources.

$$\text{Estmtd Test Time Left[Release]} = \text{Estimated Test Left[Release]} / \text{Testing Resources} / \text{Test Resource Productivity}$$

Units: Month

Estimated testing time left.

$$\text{Estmtd Tests to Be Designed[Release]} = \text{Features to Design[Release]} * \text{Test Coverage} / \text{Code Covered by Test} * \text{Modules for a feature}$$

Units: Test

The number of tests to be designed.

$$\text{Expctd Code to Write After Being First[Release,Productivity Base]} = \text{Max} (0 , \text{Perceived Estimated Code to Write[Release]} - \text{Expctd Code Written Before Being First[Release,Productivity Base]})$$

Units: module

The code that is expected to be written after becoming the main release depends on how much code is left to be written and how long we expect to stay in the shadow of other projects.

Expctd Code Written Before Being First[Release,Productivity Base] = Expctd Resource Available to New Dev[Release] * Prcvd Productivity[Release,Productivity Base] * Time in Shadow[Release]

Units: module

The expected progress during the time the release is in the shadow of more eminent ones.

Expctd Resource Available to First Release New Dev = Total Dev Resources * Prcvd Frac Resource to New Dev * Prcvd Frac to Current Release * SUM ("If Ri Nth Release?"[Release!,R1] * Expected Fraction to This Release[Release!])

Units: Person

The resources available for the first (highest priority/most eminent) release is calculated from looking at the current release with the highest priority.

Expctd Resource Available to New Dev[Release] = Total Dev Resources * Prcvd Frac Resource to New Dev * Prcvd Frac to Current Release * Expected Fraction to This Release[Release]

Units: Person

From total resources, only the fraction that is not going to CE, Rework, or other releases, can be expected to be assigned to this release.

Expected Fraction to This Release[Release] = SUM (Perceived Fraction to Nth Release[NthRelease!] * "If Ri Nth Release?"[Release,NthRelease!])

Units: Dmnl

The expected fraction of resources going to each release depends on where in the ranking that release is perceived to be.

Frac Resources to New Dev = XIDZ (SUM (Development[Release!]) , SUM (Development[Release!] + Rework[Release!]) , 0.9)

Units: Dmnl

Fraction of resources going to development

Frac Resources to Nth Release[NthRelease] = SUM (Fraction of Desired CR Allocated[Release!] * "If Ri Nth Release?"[Release!,NthRelease])

Units: Dmnl

The fraction of desired resources going to the Nth active release. Alternatively we could look at the fraction of resources actually going, but that will come up with a too conservative estimate for what will be left for each project.

Frac Time Cut = 1

Units: Dmnl [0,1.5] Fraction of time requested that is normally given to the team.

Fraction of Resources to Current Release = $XIDZ (\text{Allctd Ttl Dev Resource} , \text{Allctd Ttl Dev Resource} + \text{Alloctd Ttl CE Resource} , \text{Initial Prcvd Fraction to Current release})$
Units: Dmnl

The fraction of resources allocated to current releases.

Fraction Total Tests Passed[Release] = $ZIDZ (\text{Test Ran[Release]} , \text{Total Tests for This Release[Release]})$

Units: Dmnl

Fraction Total Tests Passed.

"Frctn Test-Dev Overlap" = 0.1

Units: Dmnl

The fraction of test overlap with development is indeed not constant, but changing depending on where in the process we are. So when all the development is done, there is no overlap, while there is much more overlap at the beginning. The small fraction makes the constancy assumption OK.

"If Ri Nth Release?"[Release,NthRelease] = if then else (Sorted Vector of Releases[NthRelease] + 1 = Release, 1, 0) * (1 - "Release Ready?"[Release]) * "Release Started?"[Release]

Units: Dmnl

This equation finds out if the each release is the Nth active release. For example, if first release is GA then the second release will be 1st, and so on. Note that releases which have not started or have finished are not counted.

Initial Prcvd Fraction to Current release = 0.9

Units: Dmnl

Initially people come with the perception that only 10% of time need be spent on current engineering

Negotiated Release Date[Release] = (Reqstd Release Date[Release] - Time) * Frac
Time Cut + Time

Units: Month

The release date given to the team, after negotiations.

Normal Design Time = 2

Units: Month

normal length of the design phase.

Normal Threshold for Replan = 2

Units: Month

Number Replans So Far[Release] = (SQRT ((2 + Strength of Threshold Feedback) ^ 2 + 8 * Accumulated Change in Release Date[Release] / Normal Threshold for Replan) - (2 + Strength of Threshold Feedback)) / (2 * Strength of Threshold Feedback) - Modulo ((SQRT ((2 + Strength of Threshold Feedback) ^ 2 + 8 * Accumulated

Change in Release Date[Release] / Normal Threshold for Replan) - (2 + Strength of Threshold Feedback)) / (2 * Strength of Threshold Feedback) , 1)

Units: Dmnl

Number of replans can be tracked based on the accumulated change in the release date and how the management changes the formal schedule based on that accumulated informal change. The formula comes from the assumption that threshold for changing release date is $B \cdot (1 + a \cdot n)$, where B is threshold for the first replan, a is the strength parameter and n is the replan numbers so far. By summing this over $n=1..N$ we get to the total delay corresponding to N replans, and by finding N in terms of that summation, we can find the Time equivalent. This variable, however, is not changing anything at the current formulation, in the other words, it is assumed that official replans are not so significant in determining the pressure, rather, the perception of the work status and what will become the official replan in future are important. This makes the behavior smoother (since the discrete replans are removed) but the overall dynamics don't change.

Perceived Estimated Code to Write[Release] = INTEG(Chng Prcvd Estmtd Code[Release] , Estmtd Code to Write[Release])

Units: module

Perceived code to be written based on current estimates.

Perceived Fraction to Nth Release[NthRelease] = INTEG((Frac Resources to Nth Release[NthRelease] - Perceived Fraction to Nth Release[NthRelease]) / Time to Perceive Resource Availability , XIDZ (1, SUM ((1 - "Release Ready?"[NthRelease!]) * "Release Started?"[NthRelease!]) , 1))

Units: Dmnl

The fraction of resources given to the Nth release is perceived as we go forward. The initial value is set to assume equal allocation between all active releases.

Prct Time Allocated[Release] = XIDZ (Negotiated Release Date[Release] - Time , Reqstd Release Date[Release] - Time , 1)

Units: Dmnl

Fraction of time requested, given to the development team.

Prcvd Frac Resource to New Dev = INTEG((Frac Resources to New Dev - Prcvd Frac Resource to New Dev) / Time to Perceive Resource Availability , 0.9)

Units: Dmnl

It takes time for people to learn what percentage of resources are really available for new development, vs. rework and current engineering. The perception is assumed to aggregate across multiple releases since people interact and see what happens across releases. At the beginning this perception is assumed to equal 0.9

Prcvd Frac to Current Release = INTEG((Fraction of Resources to Current Release - Prcvd Frac to Current Release) / Time to Perceive Resource Availability , Initial Prcvd Fraction to Current release)

Units: Dmnl

People adjust their perception about what fraction of work is given to current release

Prvcd Productivity[Release,Current] = INTEG((Productivity[Release] - Prvcd Productivity[Release,Current]) / Time to Perceive Productivity , Normal Productivity[Release])

Prvcd Productivity[Release,Normal] = INTEG(0 * (Productivity[Release] - Prvcd Productivity[Release,Current]) / Time to Perceive Productivity , Normal Productivity[Release])

Units: module/(Month*Person)

Perceived productivity slowly adjust to real productivity observed on the field.

Project Readiness Threshold = 0.95

Units: Dmnl

The pass rate necessary for release

Rank Release[Release] = (Total Number of Releases - Release + 1) * (1 - "Release Ready?"[Release]) * "Release Started?"[Release]

Units: Dmnl

Gives the rank of each release: highest number for the first release, 0 for inactive ones.

Release Date Planning[Release] = if then else (Estmtd Dev Time Left[Release ,Current] > 0 :AND: Sanctioned Release Date[Release] = 0, Allocated Release Date[Release] / TIME STEP , if then else (Estmtd Dev Time Left[Release ,Current] > 0, 1, 0) * (Allocated Release Date[Release] - Sanctioned Release Date[Release]) / Time to Replan)

Units: Dmnl

The formulation makes sure that we start with initial scheduled release date and then adjust it to the allocated one slowly.

"Release Ready this Step?"[Release] = if then else (Fraction Total Tests Passed[Release] >= Project Readiness Threshold , 1, 0) * (1 - "Release Ready?"[Release])

Units: Dmnl

The flag that becomes 1 at the time the release gets ready and then stays at 0.

"Release Ready?"[Release] = INTEG(Chng Status[Release] , 0)

Units: Dmnl

The counter variable showing if the release is ready or not

"Release Started?"[Release] = if then else (Release Start Date[Release] > Time , 0, 1)

Units: Dmnl

Gives 1 if the release has started, otherwise 0.

Reqstd Release Date[Release] = Time + Time Reqstd for Des Dev Test Left[Release]

Units: Month

Requested release date by development organization.

Sanctioned Release Date[Release] = INTEG(Release Date Planning[Release] , 0)

Units: Month

Sorted Vector of Releases[Release] = VECTOR SORT ORDER (Rank
Release[Release] , -1)

Units: Dmnl

This variable sorts and finds out the rank of each release among the active releases. So the first active release gets the rank 0 and all the finished releases get the rank 5 (the number of releases).

Strength of Threshold Feedback = 0.5

Units: Dmnl

This is the power to which the number of replans are raised, in order to determine how larger the Threshold gets for replanning. A value of 1 means for the 2nd replan we have 2 times of normal Threshold and for the 3ed one 3 times, so on. A value of 0.5 suggests that for 2nd replan we have $\sqrt{2}=1.41$ times longer Threshold etc.

SW Effect Replan on Replan Threshold = 1

Units: Dmnl

Put 1 to have the effect of number of replans on Threshold for a new replan. Put 0 for not having this feedback on.

Time in Shadow[Later Releases] = Max (0, if then else ("If Ri Nth Release?"[Later
Releases,R1] = 1, 0, VECTOR ELM MAP (Sanctioned Release Date[R1] , Later
Releases - 2) - Time))

Time in Shadow[R1] = 0

Units: Month

The time that a release is in the shadow of more important releases is as long as there is a more urgent release out there.

Time Reqstd for Des Dev Test Left[Release] = Estmtd Design Time Left[Release
] * (1 - "Frctn Des-Dev Overlap") + Estmtd Dev Time Left[Release,Current] + Estmtd
Test Time Left[Release] * (1 - "Frctn Test-Dev Overlap")

Units: Month

The time required for the rest of the project depends on how much time testing, development, and design need all together.

Time to Perceive Current Status = 2

Units: Month

There is a delay in assessing and perceiving the current status of project. This delay is important in how fast we plan and react to the state of project.

Time to Perceive Productivity = 10

Units: Month

Time to Perceive Productivity

Time to Perceive Resource Availability = 50

Units: Month
Time to Perceive Resource Availability

Total Number of Releases = ELMCOUNT(Release)

Units: Dmnl
The total number of releases modeled

Total Tests for This Release[Release] = Estmtd Tests to Be Designed[Release
] + Total Tests to Cover[Release]

Units: Test
Total tests for the current release.

Control: Simulation Control Parameters

FINAL TIME = 100

Units: Month
The final time for the simulation.

INITIAL TIME = 0

Units: Month
The initial time for the simulation.

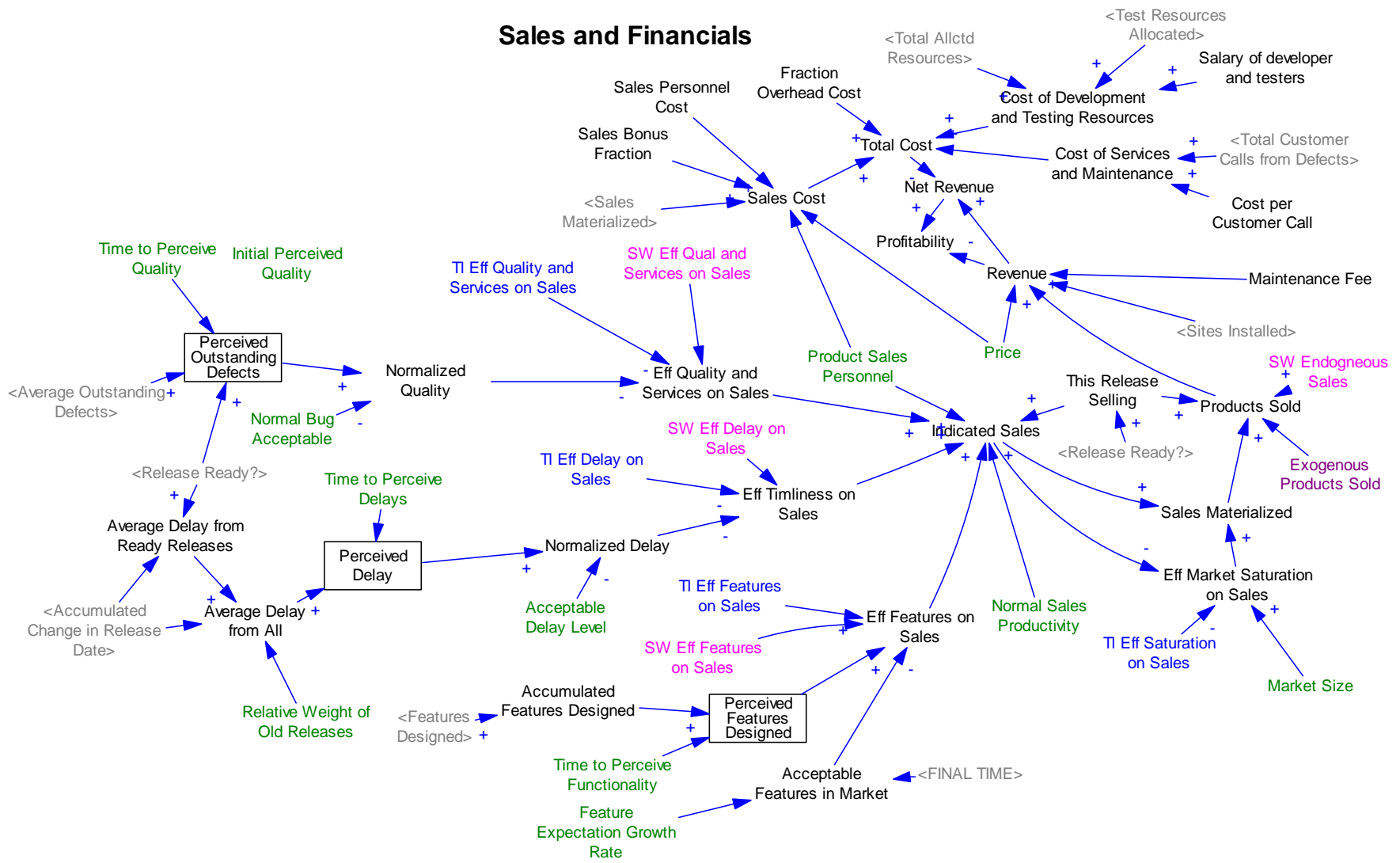
SAVEPER = 1

Units: Month
The frequency with which output is stored.

TIME STEP = 0.03125

Units: Month
The time step for the simulation.

Sales and Financials



Sales and Financial

Acceptable Delay Level = 4

Units: Month

The normal level of delay is put at 4 month, suggesting that if we have delays that long we get normal sales.

Acceptable Features in Market = RAMP (Feature Expectation Growth Rate , 20 , FINAL TIME)

Units: feature

This is the number of new features that need to be in a new release to make that release reasonable in the eyes of customer.

Accumulated Features Designed[R1] = Features Designed[R1]

Accumulated Features Designed[Later Releases] = VECTOR ELM MAP (Accumulated Features Designed[R1] , Later Releases - 2) + Features Designed[Later Releases]

Units: feature

This is the total number of features from the first release to this one.

Average Delay from All[Release] = (Accumulated Change in Release Date[Release] + Relative Weight of Old Releases * Average Delay from Ready Releases) / (1 + Relative Weight of Old Releases)

Units: Month

The average delay is a weighted average of delay so far in the current release, and the delay observed in past releases.

Average Delay from Ready Releases = ZIDZ (SUM (Accumulated Change in Release Date[Release!] * "Release Ready?"[Release!]) , SUM ("Release Ready?"[Release!]))

Units: Month

Calculating the average of delay across all the ready projects.

Cost of Development and Testing Resources = Salary of developer and testers * (SUM (Test Resources Allocated[Release!]) + Total Allctd Resources)

Units: \$/Month

Total Cost of Development and Testing Resources allocated to this project. It assumes that non-allocated resources are used elsewhere and therefore not billed to this product.

Cost of Services and Maintenance = Cost per Customer Call * SUM (Total Customer Calls from Defects[Release!])

Units: \$/Month

Total Cost of Services and Maintenance, assumed to be a function of the number customer calls.

Cost per Customer Call = 200

Units: \$/(Defect*Site)

The cost of each customer call arising from a defect, in terms of service labor etc.

Eff Features on Sales[Release] = TI Eff Features on Sales (XIDZ (Perceived Features Designed[Release] , Acceptable Features in Market , 1)) * SW Eff Features on Sales + 1 - SW Eff Features on Sales

Units: Dmnl

The effect of new features on sales depends on how many features we have vs. what is the acceptable level of features we could have.

Eff Market Saturation on Sales = TI Eff Saturation on Sales (XIDZ (Market Size , SUM (Indicated Sales[Release!]) , 10))

Units: Dmnl

The effect of market saturation depends on how much sales we could potentially do, vs. what the market needs.

Eff Quality and Services on Sales = TI Eff Quality and Services on Sales (Normalized Quality) * SW Eff Qual and Services on Sales + 1 - SW Eff Qual and Services on Sales

Units: Dmnl

Effect of quality and services on sales of the product.

Eff Timeliness on Sales[Release] = TI Eff Delay on Sales (Normalized Delay[Release]) * SW Eff Delay on Sales + 1 - SW Eff Delay on Sales

Units: Dmnl

Effect of time-to-market on sales.

Exogenous Products Sold = 10

Units: Site/Month

Exogenous sales is used for model testing

Feature Expectation Growth Rate = 30

Units: feature/Month

The rate of growth in the number of features needed to remain competitive in the market place.

Fraction Overhead Cost = 0.5

Units: Dmnl

Overhead cost, as a fraction of R&D and Service labor costs.

Indicated Sales[Release] = Product Sales Personnel * Normal Sales Productivity * Eff Features on Sales[Release] * Eff Quality and Services on Sales * Eff Timeliness on Sales[Release] * This Release Selling[Release]

Units: Site/Month

The indicated sales depends on number of sales person, their productivity and how that productivity depends on quality, timeliness, and features of the product.

Initial Perceived Quality = 1

Units: Dmnl

The initial quality level (0 for very good quality, 1 for medium, and higher for poor) is what customers have in mind before observing the products real quality.

Maintenance Fee = 1000

Units: \$/(Month*Site)

The maintenance fee charged.

Market Size = 50

Units: Site/Month

This size is defined in terms of number of sites per month in a steady state, not including any of the diffusion dynamics etc. Therefore is fairly simplistic.

Net Revenue = Revenue - Total Cost

Units: \$/Month

Net revenue of the product.

Normal Bug Acceptable = 300

Units: Defect

This is the average number of bugs in a medium release.

Normal Sales Productivity = 2

Units: Site/(Month*Person) The productivity of sales personnel in selling this product.

Normalized Delay[Release] = XIDZ (Acceptable Delay Level , Perceived Delay[Release] , 5)

Units: Dmnl

Delay level normalized by some acceptable delay.

Normalized Quality = XIDZ (Normal Bug Acceptable , Perceived Outstanding Defects , 5)

Units: Dmnl

Normalized quality of the product.

Perceived Delay[Release] = DELAY1 (Average Delay from All[Release] , Time to Perceive Delays)

Units: Month

Perceived delivery delay.

Perceived Features Designed[Release] = INTEG((Accumulated Features Designed[Release] - Perceived Features Designed[Release]) / Time to Perceive Functionality , Accumulated Features Designed[Release])

Units: feature

Perceived number of the features included in each release.

Perceived Outstanding Defects = INTEG((Average Outstanding Defects - Perceived Outstanding Defects) / Time to Perceive Quality * "Release Ready?"[R1] , Initial Perceived Quality * Normal Bug Acceptable)

Units: Defect

The initial perceived quality sets the expectations from the company before the release is out. Later on it is adjusted to observed quality, as soon as first release is out.

Price = 250000

Units: \$/Site The price of each installation of the software.

Product Sales Personnel = 5

Units: Person

The number of dedicated sales personnel for this product.

Products Sold[Release] = Exogenous Products Sold * This Release Selling[Release] * (1 - SW Endogenous Sales) + SW Endogenous Sales * Sales Materialized[Release]

Units: Site/Month

The number of products sold and sites installed in a month.

Profitability = ZIDZ (Net Revenue , Revenue)

Units: Dmnl

Profitability of the product line.

Relative Weight of Old Releases = 0.5

Units: Dmnl

The weight of old releases in comparison with the focal release, when determining the effect of delay on sales.

Revenue = SUM (Products Sold[Release!] * Price) + Maintenance Fee * SUM (Sites Installed[Release!])

Units: \$/Month

The revenue earned from this product line.

Salary of developer and testers = 6000

Units: \$/(Month*Person) Normal salary of developer and testers

Sales Bonus Fraction = 0.01

Units: Dmnl

The fraction of sales that goes as sales bonus

Sales Cost = Product Sales Personnel * Sales Personnel Cost + SUM (Sales Materialized[Release!]) * Sales Bonus Fraction * Price

Units: \$/Month

The cost of salesforce.

Sales Materialized[Release] = Indicated Sales[Release] * Eff Market Saturation on Sales

Units: Site/Month

The real sales materialized not only depends on the potential level, but on market availability.

Sales Personnel Cost = 50000

Units: \$/(Month*Person) The fixed salary of sales personnel

SW Eff Delay on Sales = 1

Units: Dmnl

Put 1 to have effect of schedule delay on attractiveness, put 0 not to include that effect.

SW Eff Features on Sales = 1

Units: Dmnl

Put 1 to have the effect of features on sales, put 0 to eliminate that effect.

SW Eff Qual and Services on Sales = 1

Units: Dmnl

Put 1 to have effect of quality on sales, put 0 not to include that effect.

SW Endogenous Sales = 1

Units: Dmnl

Put 1 to have endogenous sales rate. Put 0 to get the fixed, exogenous rate.

This Release Selling[Early Releases] = "Release Ready?"[Early Releases] * (1 - VECTOR ELM MAP ("Release Ready?"[R1] , Early Releases))

This Release Selling[R6] = "Release Ready?"[R6]

Units: Dmnl

Only the latest release is selling at any time.

Time to Perceive Delays = 5

Units: Month

Time to perceive the delays by customers.

Time to Perceive Functionality = 6

Units: Month

Time to perceive functionality of the software.

Time to Perceive Quality = 15

Units: Month

Time to perceive the quality of the product.

TI Eff Delay on Sales ([(0,0)-(1,1.5)],(0,0),(0.195719,0.203947),(0.321101,0.447368)
,(0.455657,0.690789),(0.666667,0.888158),(1,1),(2,1.15),(4,1.3))

Units: Dmnl

Table for effect of delays on sales.

TI Eff Features on Sales ([(0,0)-(2,1.5)],(0,0),(0.311927,0.0263158),(0.489297,0.118421)
,(0.66055,0.355263),(0.795107,0.598684),(0.880734,0.815789),(1,1)
,(1.24159,1.18421),(1.5,1.3),(2,1.4))

Units: Dmnl

Table for effect of feature availability on sales.

TI Eff Quality and Services on Sales ([(0,0)-(4,2)],(0,0),(0.464832,0.587719)
,(0.685015,0.789474),(1,1),(2,1.3),(4,1.5))

Units: Dmnl

Table for effect of quality on sales of the product.

TI Eff Saturation on Sales ([(0,0)-(3,1)],(0,0),(0.4,0.4),(0.706422,0.583333)
,(1,0.7),(1.45872,0.837719),(2.07339,0.934211),(3,1))

Units: Dmnl

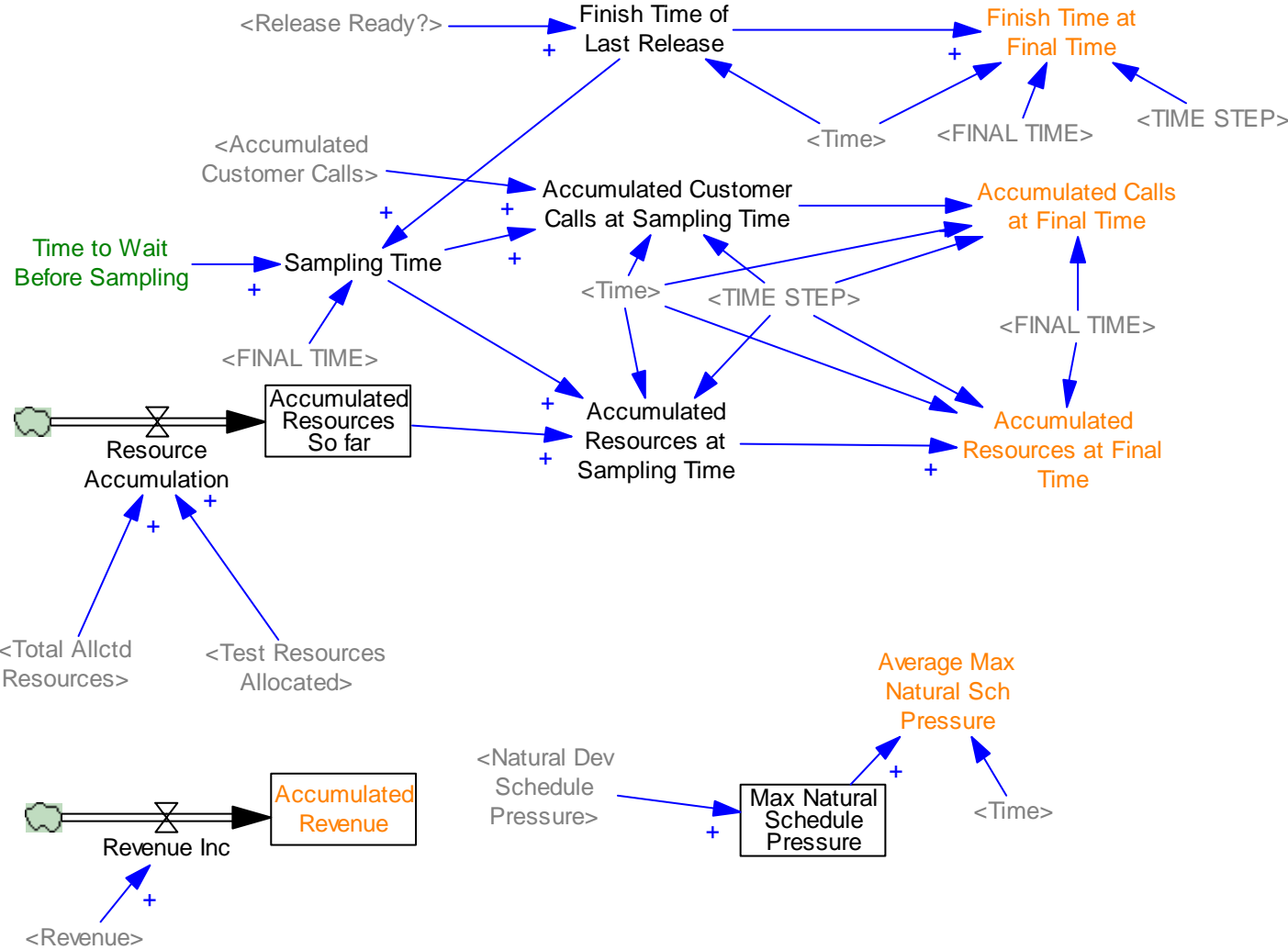
There is a market saturation effect, i.e. sales can not grow more than a limit in the market.

Total Cost = (Cost of Development and Testing Resources + Cost of Services and Maintenance) * (1 + Fraction Overhead Cost) + Sales Cost

Units: \$/Month

Total R&D, Service, and salesforce costs of the project.

Metrics



Metrics

Accumulated Calls at Final Time = if then else (Time <= FINAL TIME - TIME STEP , 0, Accumulated Customer Calls at Sampling Time / TIME STEP)

Units: Defect*Site/Month

Accumulated customer calls at final time for payoff function in optimization.

Accumulated Customer Calls at Sampling Time = SAMPLE IF TRUE(Time <= Sampling Time :AND: Time + TIME STEP > Sampling Time , Accumulated Customer Calls , 0)

Units: Defect*Site

The accumulated number of customer calls at the time of the sampling.

Accumulated Resources at Final Time = if then else (Time <= FINAL TIME - TIME STEP , 0, Accumulated Resources at Sampling Time / TIME STEP)

Units: Person

accumulation resources at final time for optimization purposes.

Accumulated Resources at Sampling Time = SAMPLE IF TRUE(Time <= Sampling Time :AND: Time + TIME STEP > Sampling Time , Accumulated Resources So far , 0)

Units: Month*Person

Picks up the value of accumulated resources at the sampling time.

Accumulated Resources So far = INTEG(Resource Accumulation , 0)

Units: Month*Person

Total number of people involved in development and testing of the product.

Accumulated Revenue = INTEG(Revenue Inc , 0)

Units: \$ The accumulated revenue through time.

Average Max Natural Sch Pressure = ZIDZ (Max Natural Schedule Pressure , Time)

Units: Dmnl

Average of the maximum schedule pressure among different releases, through time.

Finish Time at Final Time = if then else (Time <= FINAL TIME - TIME STEP , 0, Finish Time of Last Release / TIME STEP)

Units: Dmnl

Finish time of 6 releases at the end, divided by time step to make sure the payoff is equal to final time, in optimization control.

Finish Time of Last Release = SAMPLE IF TRUE("Release Ready?"[R6] > 0 :AND: Finish Time of Last Release = 0, Time , 0)

Units: Month

This picks up the finish time for the last release.

Max Natural Schedule Pressure = INTEG(Min (3, VMAX (Natural Dev Schedule Pressure[Release!])) , 0)

Units: Month

The maximum of schedule pressure across different releases.

Resource Accumulation = SUM (Test Resources Allocated[Release!]) + Total Allctd Resources

Units: Person

Sum of resources actively involved in the project at this time.

Revenue Inc = Revenue

Units: \$/Month

The increase rate in revenue.

Sampling Time = if then else (Finish Time of Last Release > 0, Min (Finish Time of Last Release + Time to Wait Before Sampling , FINAL TIME) , FINAL TIME)

Units: Month

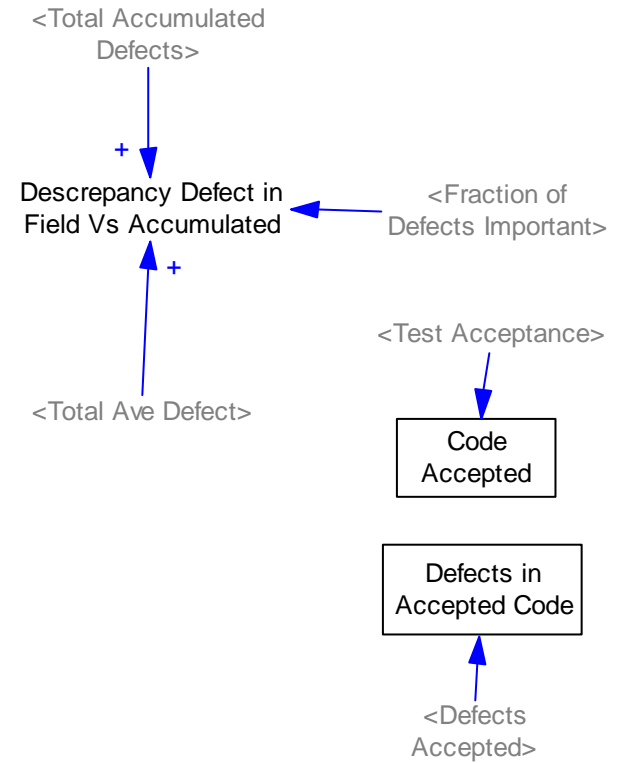
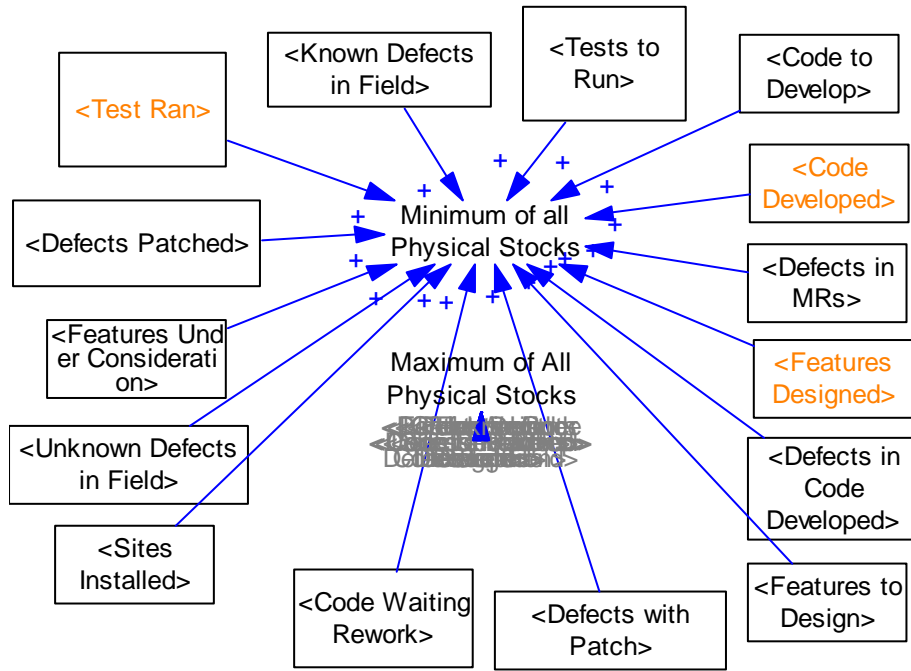
Sampling time for observing value of important metrics.

Time to Wait Before Sampling = 10

Units: Month

There is no sampling in the first few months to avoid picking up things before project are under way.

Testing of Model



Productivity Base: Current, Normal

This subscript is to simplify equations written in parallel, but based on two different productivity levels, one is the current level of the productivity, and another is based on the normal level of productivity. Current productivity is what is really used in most of the planning of the work, while the normal productivity is what is driving how fast people work, how is their quality and so on.

Release : R1, Later Releases

Different releases for the same product

SRelease <-> Release

Alphabetical list of model variables and their groupings:

Variable	Group/View Name
Acceptable Delay Level	Sales and Financial
Acceptable Features in Market	Sales and Financial
Accmltd Code Accepted	Architecture and Code Base Quality
Accmltd Features Designed	Architecture and Code Base Quality
Accumulated Calls at Final Time	Metrics
Accumulated Change in Release Date	Scheduling
Accumulated Customer Calls	Patches and Current Engineering
Accumulated Customer Calls at Sampling Time	Metrics
Accumulated Defects in Code	Architecture and Code Base Quality
Accumulated Features Designed	Sales and Financial
Accumulated Problems in Architecture and Design	Architecture and Code Base Quality
Accumulated Resources at Final Time	Metrics
Accumulated Resources at Sampling Time	Metrics
Accumulated Resources So far	Metrics
Accumulated Revenue	Metrics
Accumulated Total Work	Patches and Current Engineering
Allctd Ttl Dev Resource	Patches and Current Engineering
Allocated Design Finish Date	Scheduling
Allocated Design Time Left	Development
Allocated Dev Finish Date	Scheduling
Allocated Dev Time Left	Scheduling
Allocated Release Date	Scheduling
Allocated Test Time Left	Scheduling
Alloctd Ttl CE Resource	Patches and Current Engineering
Architecture Redesign	Architecture and Code Base Quality
Ave Def with Patch	Defects in the Field
Ave Dfct Dnsty in Acptd Code	Development
Ave Dfct in Rwrk Code	Development
Ave Dfct in Tstd Code	Development
Ave Known Def in Site	Defects in the Field
Ave New Problem per Cancellation	Concept and Design
Ave Prblm in Designed Feature	Concept and Design
Ave Problem in Features to Design	Concept and Design
Ave Unknown Def in Site	Defects in the Field
Average Defects Patched	Defects in the Field
Average Delay from All	Sales and Financial
Average Delay from Ready Releases	Sales and Financial
Average Max Natural Sch Pressure	Metrics
Average Outstanding Defects	Defects in the Field
Average Time in the Field	Defects in the Field
Bug Fix Approval	Fixing Bugs
Bug Fixes Designed	Development

Bug Fixes Planned	Fixing Bugs
Bugs to Fix in Future Releases	Fixing Bugs
Calls from Defects With Patch	Defects in the Field
Calls from Known Defects	Defects in the Field
Calls from Unknown Defects	Defects in the Field
Cancel Bug Fixes	Fixing Bugs
Chang in Release Date	Scheduling
Chng Prcvd Estmtd Code	Scheduling
Chng Start Date	Concept and Design
Chng Status	Scheduling
Code Accepted	Development
Code Cancellation	Development
Code Covered by Test	Development
Code Developed	Development
Code Fixes	Architecture and Code Base Quality
Code Passing Test	Development
Code to Develop	Development
Code to Fix Bugs	Fixing Bugs
Code Waiting Rework	Development
Cost of Development and Testing Resources	Sales and Financial
Cost of Services and Maintenance	Sales and Financial
Cost per Customer Call	Sales and Financial
CR Dev Resources Left After Allocation	Patches and Current Engineering
Current Dev Schedule Pressure	Development
Current Scope of Release	Concept and Design
Defect Covering by Patch	Defects in the Field
Defect Intro in Coherence of Feature Set	Concept and Design
Defect not Patching	Defects in the Field
Defect Patching	Defects in the Field
Defect per Patch	Fixing Bugs
Defect Show up	Defects in the Field
Defects Accepted	Development
Defects Fixed	Development
Defects found	Development
Defects from Dev	Development
Defects Generated	Development
Defects in Accepted Code	Development
Defects in Code Developed	Development
Defects in MRs	Development
Defects Patched	Defects in the Field
Defects Retired	Defects in the Field
Defects with Patch	Defects in the Field
Defects with Patch Retired	Defects in the Field
Demand for Test Resources	Test
Density of Defect in Code	Architecture and Code Base Quality

Density of Problems in Architecture	Architecture and Code Base Quality
Des Time Change	Concept and Design
Discrepancy Defect in Field Vs Accumulated	Testing Model
Design	Concept and Design
Design Rate	Development
Design Time Left	Concept and Design
Desired Bug Fixes for Next Release	Fixing Bugs
Desired Development Rate	Development
Desired Patch Rate	Patches and Current Engineering
Desired Resources to Current Engineering	Patches and Current Engineering
Desired Total Dev Resources	Patches and Current Engineering
Dev Reschedule	Development
Dev Resources to New Dev	Development
Dev Resources to Rework	Development
Development	Development
Development and CE Rate	Patches and Current Engineering
Development Resource to Current Engineering	Patches and Current Engineering
Development Resources to Current Release	Patches and Current Engineering
Development Time Left	Development
Dfct Dnsty in Accepted Code	Development
Dfct Dnsty in Rejctd Code	Development
Dfct Dnsty in Untstd Code	Development
Dfct from Rwrk	Development
Dfcts Left in RWrk	Development
Dsrd Dev Resources	Development
Dsrd New Dev Resources	Development
Dsrd Rwrk Rate	Development
Dsrd Rwrk Resources	Development
Dsrd Time to Do Rwrk	Development
Early Releases	Subscript
Eff Archtcr Complexity on Productivity	Development
Eff Bug Fixing on Scope of New Release	Concept and Design
Eff Features on Sales	Sales and Financial
Eff Market Saturation on Sales	Sales and Financial
Eff Old Archtctr on Design Error	Concept and Design
Eff Old Code & Archtctr on Error	Development
Eff Pressure on Error Rate	Development
Eff Pressure on Productivity	Development
Eff Profitability on Resources Allocated	Patches and Current Engineering
Eff Quality and Services on Sales	Sales and Financial
Eff Timeliness on Sales	Sales and Financial
Endogenous Release Start Date	Concept and Design
Error per Design Problem in Features	Development
Error Rate in Feature Design	Concept and Design
Error Rate in Late Features	Concept and Design

Error Rate in Original Feature Selection	Concept and Design
Error Rate New Dev	Development
Errors from Coding	Development
Errors from Design and Architecture	Development
Estimated Test Left	Scheduling
Estmtd Code to Write	Scheduling
Estmtd Design Time Left	Scheduling
Estmtd Dev Time Left	Scheduling
Estmtd Test Time Left	Scheduling
Estmtd Tests to Be Designed	Scheduling
Exogenous Products Sold	Sales and Financial
Exogenous Release Start Date	Concept and Design
Expctd Code to Write After Being First	Scheduling
Expctd Code Written Before Being First	Scheduling
Expctd Resource Available to First Release New Dev	Scheduling
Expctd Resource Available to New Dev	Scheduling
Expected Fraction to This Release	Scheduling
Feasible New Dev Rate	Development
Feasible Rwrk Rate	Development
Feature Cancellation	Concept and Design
Feature Expectation Growth Rate	Sales and Financial
Feature Requests	Concept and Design
Features Approved	Concept and Design
Features Designed	Concept and Design
Features to Design	Concept and Design
Features Under Consideration	Concept and Design
FINAL TIME	Control
Finish Time at Final Time	Metrics
Finish Time of Last Release	Metrics
Fixed Defects in Code	Architecture and Code Base Quality
Frac CE Allctd	Patches and Current Engineering
Frac CR Allctd	Patches and Current Engineering
Frac Errors Found in Rwrk	Development
Frac Resources to New Dev	Scheduling
Frac Resources to Nth Release	Scheduling
Frac Time Cut	Scheduling
Frac Total Tst Feasible	Test
Fraction Bugs Fixed in Next Release	Fixing Bugs
Fraction Code Accepted	Test
Fraction code Perceived Developed	Concept and Design
Fraction Code Written	Test
Fraction Defect Unknown	Defects in the Field
Fraction of Calls Indicating New Defects	Patches and Current Engineering
Fraction of Defects Important	Defects in the Field

Fraction of Desired CR Allocated	Patches and Current Engineering
Fraction of Resources to Current Release	Scheduling
Fraction of sites Installing All patches for a Call	Defects in the Field
Fraction Overhead Cost	Sales and Financial
Fraction Patches Installed At Sales	Defects in the Field
Fraction Sites Installing	Defects in the Field
Fraction Total Tests Passed	Scheduling
Frctn Code Cancellation	Development
Frctn Des-Dev Overlap	Development
Frctn Test-Dev Overlap	Scheduling
Frctn Tst Running	Test
Honey-Moon Period	Patches and Current Engineering
If Ri Nth Release?	Scheduling
Inc Customer Calls	Patches and Current Engineering
Inc Defect in Underlying Code	Architecture and Code Base Quality
Inc Old Design	Architecture and Code Base Quality
Inc Prblm in Old Design	Architecture and Code Base Quality
Inc Underlying Code	Architecture and Code Base Quality
Increase in Known Bugs	Fixing Bugs
Indicated Sales	Sales and Financial
Initial Features Under Consideration	Concept and Design
Initial Perceived Quality	Sales and Financial
Initial Prcvd Fraction to Current release	Scheduling
INITIAL TIME	Control
Installing Known Patches	Defects in the Field
Known Defects in Field	Defects in the Field
Known Defects Retired	Defects in the Field
Known Defects Sold	Defects in the Field
Late Feature Error Coefficient	Concept and Design
Late Feature Introduction	Concept and Design
Late Feature Problems	Concept and Design
Late Features	Concept and Design
Late Intro Time	Concept and Design
Later Releases	Subscript
Maintenance Fee	Sales and Financial
Market Size	Sales and Financial
Max Natural Schedule Pressure	Metrics
Maximum Cancellation Fraction	Development
Maximum of All Physical Stocks	Testing Model
Min Des Time	Concept and Design
Min Dev Time	Development
Min Tst Time	Test
Minimum Code Size With an Effect	Development
Minimum of all Physical Stocks	Testing Model
Modules Equivalent to Patch	Fixing Bugs

Modules for a feature	Development
Multiplier Fixing work vs. Patch	Fixing Bugs
Natural Dev Schedule Pressure	Development
Negotiated Release Date	Scheduling
Net Revenue	Sales and Financial
New Error Rate in Rwrk	Development
Normal Bug Acceptable	Sales and Financial
Normal Design Time	Scheduling
Normal Error Rate in Feature Design	Concept and Design
Normal Error Rate in Rwrk	Development
Normal Errors from Coding	Development
Normal Productivity	Development
Normal Sales Productivity	Sales and Financial
Normal Scope of New Release	Concept and Design
Normal Threshold for Replan	Scheduling
Normalized Delay	Sales and Financial
Normalized Quality	Sales and Financial
NthRelease	Subscript
Number Replans So Far	Scheduling
Ok to Plan?	Concept and Design
Patch Developed	Patches and Current Engineering
Patch Development	Patches and Current Engineering
Patch to Develop	Patches and Current Engineering
Patched Defects Sold	Defects in the Field
Patches Retired	Defects in the Field
Perceived Delay	Sales and Financial
Perceived Estimated Code to Write	Scheduling
Perceived Features Designed	Sales and Financial
Perceived Fraction to Nth Release	Scheduling
Perceived Outstanding Defects	Sales and Financial
Perceived Profitability	Patches and Current Engineering
Perceived Total Code	Concept and Design
Prnt Time Allocated	Scheduling
Prvd Frac Resource to New Dev	Scheduling
Prvd Frac to Current Release	Scheduling
Prvd Productivity	Scheduling
Price	Sales and Financial
Problem Cancellation	Concept and Design
Problems from Cancellation	Concept and Design
Problems in Approved Features	Concept and Design
Problems in Designed Features	Concept and Design
Problems Introduced in Design	Concept and Design
Problems Moved to Design	Concept and Design
Problems Per Feature Interaction	Concept and Design
Product Sales Personnel	Sales and Financial

Productivity	Development
Productivity Base	Subscript
Productivity in Writing Patch	Patches and Current Engineering
Products Retired	Defects in the Field
Products Sold	Sales and Financial
Profitability	Sales and Financial
Project Readiness Threshold	Scheduling
Rank Release	Scheduling
Relative Importance of Architecture Problems	Architecture and Code Base Quality
Relative Size of Expected Bug Fixing	Concept and Design
Relative Weight of Old Releases	Sales and Financial
Release	Subscript
Release Date Planning	Scheduling
Release Ready this Step?	Scheduling
Release Ready?	Scheduling
Release Start Date	Concept and Design
Release Started?	Scheduling
Reqstd Release Date	Scheduling
Resource Accumulation	Metrics
Retest Required	Test
Revenue	Sales and Financial
Revenue Inc	Metrics
Rework	Development
Salary of developer and testers	Sales and Financial
Sales Bonus Fraction	Sales and Financial
Sales Cost	Sales and Financial
Sales Materialized	Sales and Financial
Sales Personnel Cost	Sales and Financial
Sampling Time	Metrics
Sanctioned Code Complete	Development
Sanctioned Design Complete	Concept and Design
Sanctioned Release Date	Scheduling
SAVEPER	Control
Scope of New Release	Concept and Design
Sites Installed	Defects in the Field
Sites Reporting One Defect	Defects in the Field
Size of Next Bug Fix	Fixing Bugs
Sorted Vector of Releases	Scheduling
SRelease	Subscript
Strength of Complexity on Productivity Effect	Development
Strength of Threshold Feedback	Scheduling
SW Eff Archtcr Complexity on Productivity	Development
SW Eff Bug Fix on Scope	Concept and Design
SW Eff Delay on Sales	Sales and Financial
SW Eff Features on Sales	Sales and Financial

SW Eff Old Arch on Design Problems	Concept and Design
SW Eff Old Code and Archtctr on Error	Development
SW Eff Pressure on Error	Development
SW Eff Pressure on Productivity	Development
SW Eff Profit on Resources	Patches and Current Engineering
SW Eff Qual and Services on Sales	Sales and Financial
SW Effect Replan on Replan Threshold	Scheduling
SW Endogenous Start	Concept and Design
SW Endogenous Sales	Sales and Financial
SW Exogenous CE Resources	Patches and Current Engineering
SW Fixing Bugs	Fixing Bugs
SW Higher Priority to Current	Patches and Current Engineering
Test Acceptance	Development
Test Coverage	Test
Test Design	Test
Test Feasible	Test
Test Ran	Test
Test Rate	Test
Test Rejection	Development
Test Resource Productivity	Test
Test Resources Allocated	Test
Testing Resources	Test
Tests to Run	Test
This Release Selling	Sales and Financial
Threshold for New Project Start	Concept and Design
Time in Shadow	Scheduling
Time Reqstd for Des Dev Test Left	Scheduling
Time Since Release Ready	Patches and Current Engineering
TIME STEP	Concept and Design
Time to Adjust Resources	Patches and Current Engineering
Time to Cancel Bug Fixes	Fixing Bugs
Time to Patch	Patches and Current Engineering
Time to Perceive Current Status	Scheduling
Time to Perceive Delays	Sales and Financial
Time to Perceive Functionality	Sales and Financial
Time to Perceive Productivity	Scheduling
Time to Perceive Profitability	Patches and Current Engineering
Time to Perceive Quality	Sales and Financial
Time to Perceive Resource Availability	Scheduling
Time to Replan	Concept and Design
Time to Show up for Def with Patch	Defects in the Field
Time to Show Up for Defects	Defects in the Field
Time to Wait Before Sampling	Metrics
TI Eff Bug Fix on Scope	Concept and Design
TI Eff Delay on Sales	Sales and Financial

TI Eff Features on Sales	Sales and Financial
TI Eff Old Arch on Design Problems	Concept and Design
TI Eff Old Code & Archtctr on Error	Development
TI Eff Pressure on Error Rate	Development
TI Eff Pressure on Productivity	Development
TI Eff Profit on Resources	Patches and Current Engineering
TI Eff Quality and Services on Sales	Sales and Financial
TI Eff Saturation on Sales	Sales and Financial
TI Eff Schdul Prssr on Cancellation	Development
TI Tst Dependency on Dev	Test
Total Accumulated Defects	Architecture and Code Base Quality
Total Allctd Resources	Patches and Current Engineering
Total Ave Defect	Defects in the Field
Total Code	Test
Total Cost	Sales and Financial
Total Customer Calls from Defects	Patches and Current Engineering
Total Defects in Field	Defects in the Field
Total Defects Introduced	Defects in the Field
Total Desired Resources	Patches and Current Engineering
Total Dev Resources	Patches and Current Engineering
Total Development	Development
Total Number of Releases	Scheduling
Total Tests for This Release	Scheduling
Total Tests to Cover	Test
Totl Code to Tst	Development
Totl Code Tstd	Development
Tst Effectiveness	Development
Tst Rate Feasible	Test
Tst Rate from Resources	Test
Tst Rjctn Frac	Development
Tstd Code Accptd	Development
Ttl Des Resource to CR	Patches and Current Engineering
Ttl Des Resource to CE	Patches and Current Engineering
Unknown Defects in Field	Defects in the Field
Unknown Defects Introduced	Defects in the Field
Unpatched Defects Sold with Patch	Defects in the Field
Weighted Error Density in Architecture and Code Base	Architecture and Code Base Quality