Using Simulation to Investigate IT Micro-processes

Alexey Olkov, Daniel Port
University of Hawaii at Manoa
olkov@hawaii.edu, dport@hawaii.edu

Abstract

The objective of creating a simulation toolkit SimSWE is to provide means for gaining confidence in the empirical analysis (automated or otherwise) of software micro-processes and methods for validating or obtaining evidence to support software engineering hypotheses and theory.

1. Introduction

Trying to analyze and understand the behavior of the system or the process is a very challenging task especially if it consists of multiple sub-systems or sub-processes. In order to do that, it is essential to see the cause-effect relationships between the parts. Knowledge of these relationships can be gained in by means of empirical studies and experiments. However, if we are dealing with processes that require a lot of resources (time, cost, labor), running the experiment all over until we see the connection, can be difficult, and, often, unrealistic. For the situations like that, a simulation can be a solution. The simulation toolkit SimSWE is designed to provide the means for simulation of the software development process, as a mix of various sub-processes in order to model the different trends of development.

Having a tool for simulation of software development will help the developers and project mangers in two ways: first, it can help better understand the effect of certain values of various parameters of development (late project start, often builds, inexperienced personnel etc.); second, the SimSWE can help view the project in terms of GQM (Goal Question Metrics) paradigm, which can change the way the process is measured, by reducing the number of metrics to those, that are only relevant for specific business goals.

2. GQM Paradigm

The main role of a project manager is a rigorous control of the processes and environments, and control is always defined in terms of continuous software measurement ("You cannot control what you cannot measure", T. DeMarco). Software measurement is an essential part of Software Engineering, whose scope of activities includes planning, design, cost estimation, planning, testing, etc., and is assisting a project manager by making characteristics and relationships more clear and assessing various problems, giving the information on the current status of a project, process or resources.

One of the main mechanisms of measurement is by collecting and analyzing various metrics of the project. Metrics can give information on multiple aspects of the project or a process and can be implemented in many ways. However, no matter how the metrics are being collected, it has been a ubiquitous problem when too much effort gets wasted on gathering and analyzing metrics that are not important for specific business goal, and do not provide any relevant information. In order to be confident whether the effort is spent purposefully, the intuitive, but rather powerful Goal Question Metric (GQM) approach was introduced.

The purpose of GQM is to measure something meaningful rather than the random assortment of metrics that is frequently considered. GQM makes people think about the reasons behind and information they would like to get out of metrics – before collecting metric data. This approach leads to direct relationships between low level metrics and high level goals within the appropriate context, and also helps avoid the situation when the metrics are being collected simply because they can be collected, without any connection to the business goal.

In essence, GQM approach requires three steps made by the managers or designers:

1. Set goals specific to needs in terms of purpose, perspective and environment
2. Refine the goals into quantifiable questions that are tractable
3. Deduce the metrics and data to be collected (and the means for collecting them) to answer the questions

So, the panning of data collection always comes from thinking of the goals first, and the other way around – when the data is collected, it is already known which goal it is related to through a set of questions linked to these goals. Figure 1 demonstrates the top-down approach of planning, and bottom-up process of analyzing the collected data. The level of satisfaction of goal G1 is done by answering the questions Q1 and Q2, which are done by analyzing the metric data M1 and M2.

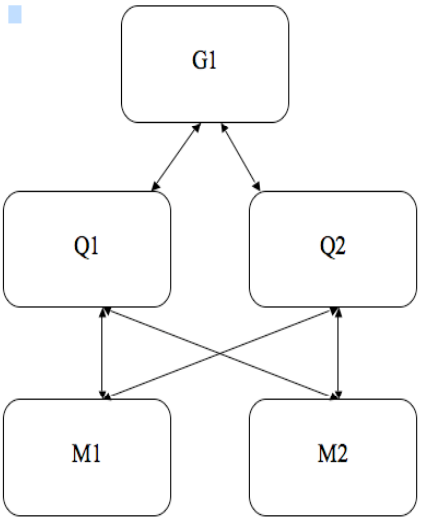

Figure 1. GWM Model for collecting and analyzing the data

Obviously, using this approach eliminates the situation when metrics are not related to any goals and are just being collected.

Even though having a GQM approach can help plan the measurement and control process, it can still be a challenging task to understand which metrics need to be collected and how they should be analyzed.

3. GQM and SimSWE

One of the applications of SimSWE is to test GQM metrics before using them on a real project in order to ensure they are measuring something meaningful from a managerial point of view, rather than a set of multiple parameters that do not give any valuable information; also, it will help us observe if the system in fact behaves the way we expect it to behave for given settings. The toolkit is not intended to fully simulate an actual process, but rather give us an opportunity to look at outcomes resulting from various parameters' setting.

The GQM paradigm will serve as the language for defining the simulation. SimSWE will simulate various metrics based on adjustable behavior models that can then be output to "Question" components, which, in turn, can output to "Goal" components. Alternatively, simulated data can be output as sensor data into automated metrics collection tools such as Hackystat for analysis there.

Defining all the simulated data in terms of GQM will help to focus only on relevant data.

3.1 Example of GQM application for simulation design: requirements prioritization

The need for the requirement prioritization is described in more details in the following sections. This part describes how to view the solution of this problem from the point of GQM.

Define goal

G1: **Analyze** various strategy prioritization strategies in order **to find** the one that produces the most effective outcome in the **environment of** high volatility, **from the point of view** of a project manager.

Establish questions

Q1: Over 1000 of simulation runs, which strategy returns the highest value (on average) at random point of project termination?

Q2: Over 1000 of simulation runs, which strategy returns the lowest cost (on average) at a random point of project termination?

Q3: Over 1000 of simulation runs, which strategy returns the largest area under the cost-value curve (see example below)?

Define metrics

M1: List of requirement costs

M2: List of requirement values

Note, that we are replacing the gathered metrics with the generated data; by this we are expanding our experiments to the variety of data, that can be hardly controlled and collected during the actual measurement process.

4. SimSWE usage examples

Obviously, the simulated data is only useful when applied in a certain context for making a certain decision. A collection of simulated data, just as a collection of gathered data does not have any

value, if analyzing it does not help us better achieve specific business goals.

Analysis of the data always needs to be specific (in terms of GQM, it should satisfy the stated goals goals, otherwise it should not even be collected or simulated). Each analyser needs to be designed and coded separately with specific goals in mind, and can be highly flexible and adjustable. The examples of analyzers and how they use the simulated data, are described below.

4.1 Requirement prioritization

All development efforts take great care in choosing what is implemented. A great deal of research and debate is directed on implementation approaches. In particular agile vs. plan-based development approaches, less attention is focused on what should be implemented when, yet this is no less important in today's complex and risky software development efforts. In this, prioritization of requirements is recognized as an essential micro-process within any development process. With high customer expectations, tight schedules, and limited resources, prioritization is used to limit the scope and deliver the most essential functionality as early as possible. It is an accepted fact that for most development efforts that not all identified requirements will be implemented. Prioritization is needed, not just so as to be able to ignore the least important requirements, but also to help the project manager to resolve conflicts, plan for staged deliveries, and make the necessary trade-offs throughout the development lifecycle.

Both plan-based and agile development approaches view prioritization as a fundamental activity but they differ in their basic strategy. A requirements prioritization strategy determines what requirements are implemented and in what sequence with respect to a strategic goal such as "minimize cost." There are many different strategies. For example, "implement the lowest cost requirements first" or "implement the highest value requirements first" and some strategies are more effective than others. The primary question of interest here is in finding an effective strategy for a given development effort.

While there is a great deal of literature on requirements prioritization, little of this addresses the issue of strategy effectiveness. Perhaps one reason for this is that, with the exception of naïve strategies (e.g. implement the requirements as they appear) all strategies rely on difficult assessments such as cost estimation, value assessment, dependency analysis, and so forth. While estimating the cost of a task is generally straightforward, it is difficult to estimate the cost of a particular requirement (cost here typically is interpreted as "effort" here, not money).

Value is generally an "intangible" not easily attributed to a particular requirement. Generally it is overall value, or the value for completed groups of requirements that represent a complete set of functionality is all that is considered. So-called "earned value" is not actual value is not reliable for prioritization purposes. Furthermore, requirements prioritization is difficult to monitor and measure "in-vitro" within actual practice. Given the above issues, and many others that we have left out, the research question we are interested in is what is a practical means for investigating the effectiveness of requirements prioritization? Controlled experiments are impractical, as is common with assessing software engineering methods (e.g. how to set up exact replications with different strategies, how to prescribe requirements volatility, etc.).

In addition to the above stated challenges in dealing with intangibles and collecting data in-vitro, experiments would require a large number data points to get convergence of effectiveness measures due to the highly variable (and uncontrollable) conditions and circumstances within any given project. Comprehensive simulation is an attractive option for investigating and providing empirical support and justification of new software engineering methods whose effectiveness measures are intangible and unobservable. Such simulations are common and accepted as evidence within the management and operations research literature where the evaluation challenges are analogous to those in software engineering.

In this work we create a simulation based on requirements theory and a detailed empirical study of requirements practices. We verify that the simulation is consistent with the theory for basic agile and plan-based requirements prioritization methods. The simulation is then used to explore properties of requirements prioritization strategies and investigate two new methods suggested by application of this theory. Strategies are compared graphically and with respect to six strategy effectiveness measures under various requirements volatility scenarios. The home-ground theory states that agile methods are most effective when requirements volatility is very high, while plan-based methods are most effective when there is relatively little requirements volatility. The theory suggests that a mixture of the two methods will

generally be more effective than either alone for typical development efforts. This study seeks to answer the question "what would a mixed agile and plan-based requirements prioritization strategy look like and how effective is it?

The simulated data and analysis of different strategies are done separately. In other words, the requirements are simulated on their own (whether the volatility is realistic or not).

Result of a single run of simulation and analysis of the different prioritization is shown in Figure 1. The winning strategy is represented by the curve above the other ones. In order to make a claim that this or that strategy is the most effective for a given set of parameters (volatility rate), it is important to run the analysis for exactly the same settings multiple times. This can give us confidence that the obtained result is, in fact, an average outcome of this or that strategy.

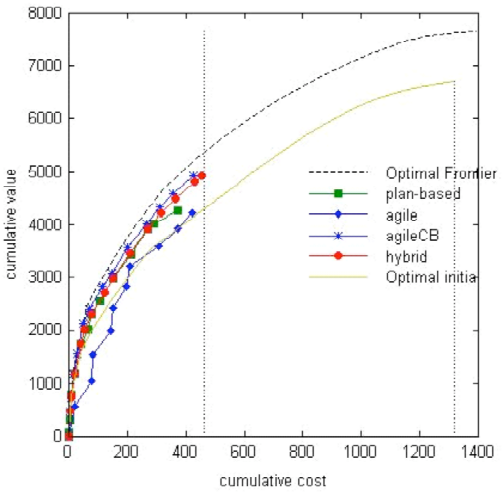

Figure 2. A sample of medium volatility run

The designing and implementing of the analyzer not only helped us to confirm our expectations regarding the strategies, but also helped understand the prioritization processes and the way the volatility affects the final outcome.

4.2 Continuous integration

Continuous Integration analyzer has not yet been implemented; however, the ways the simulation can assist in development of the analyzer is rather clear.

Continuous integration approach is a way of looking at the collected data in real-time, paying close attention to the "vital signs" of the process in order to determine if the project is "healthy".

Continuous integration concepts describe several ways of applying the analyzer, which include but are not limited to:

- An understanding of "natural variation" in healthy project vital signs for a given project, and across all projects
- An early warning system for projects in trouble.
- Additional data for project post-mortems. If the vital signs indicated health but the project died, what went wrong? What additional vital signs could have saved the patient, or did it die due to "natural causes"?

In order to be confident that the performed analysis and given recommendations are reliable, the system needs to be exhaustively tested under various circumstances. Besides, the concepts of the "healthy" project need to be strictly defined.

While "health" can be described in terms of tens of factors, for the simplicity's sake, I will give an example of simulation use for health indication from the point of view of build rate.

As the example indicates, the build analysis includes the following dimensions:

- Total number of builds
- Number of successful builds
- Number of failing builds
- Percentage successful builds

Indicators of health:

- Frequency of builds over a period of time should be reasonable
- Frequency of build failures should be low

Potential symptoms of problems:

- An abnormal number of build events
- A project with a baseline number of builds that are abnormally low or abnormally high compared to other projects
- An abnormally high number of build failures

Given these definitions, we expect that when, for example, the build rate is reasonable and failure frequency is low, the analyzer should give a "green flag" ("healthy" project). Having a simulation toolkit can help us demonstrate or test it. The data that needs to be simulated (modules) are the

number of builds, number of build failures, and the parameters for these modules should be set to low.

As mentioned before, it is not important if the parameters for, let's say, build failures does not reflect the real-life failure rate. What we want to see is that when these values are "low", the system in fact tells that the project is healthy.

The same way we can test whether the system reflects "unhealthiness" of the project appropriately – what we want to observe is that under these give high settings, the system will give a "red-flag".

Gathering the real-life data for such tests is not easily achievable, because it can require time and runs of the projects under the same parameters.

The Continuous Integration analysis consists of multiple parameters and having the simulation toolkit can help "play" with the parameters in any possible way. Other than testing of the analyzer, playing with the data parameters can assist a designer in better understanding of what a real project consists of, and how dnerdvanaifferent parameters are related to each other.

5. SimSWE structure

This section provides description of the ways the modules interact with each other, and the way the simulation runs and is controlled.

The way many of the existing simulation tools work, and manage the data flows, includes the existence of a controlling module ("Director"), and the sink of modules, possibly connected with each other. The role of the Director is to provide the way the information flow is initiated, and maintained. The Directors types differ depending on the type of simulation the user is creating.

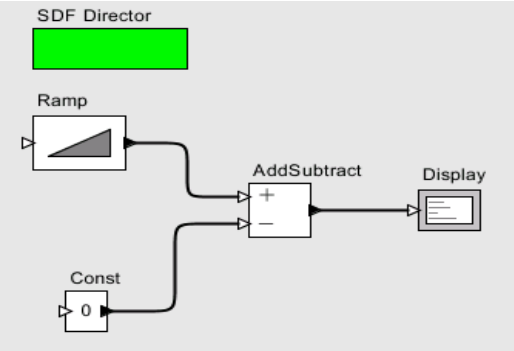

Figure 3. PTOLEMY II Graphical Tool for simulation design. Example of a simple simulation controlled by an SDF Director

Since software development process can roughly be viewed as a discrete event process, the main operation of the director would be the "time ticks" that indicate process iterations or other time intervals, at which the data is generated. The other function of the Director is to set the parameters of the environment, as opposed to the parameters of the individual activity. Such parameters may include: cost volatility, user expertise level, teamwork efficiency, IDE parameters, development language and others.

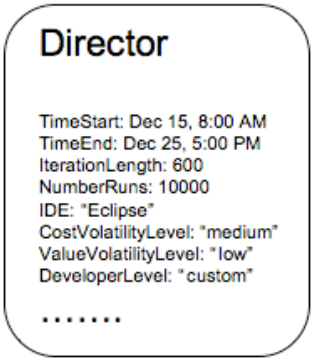

Figure 4. Example of the Director Settings

The data generated by the module can be sent to the other module as an input by setting a link between the modules, or can be redirected into an "external flow" (print out, store in the database or measurement sensor). For this purpose each link can be split into two and more lines, which duplicate the flow (i.e. such split serves as a broadcasting hub)

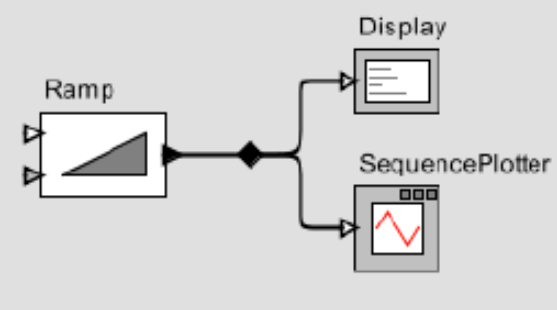

Figure 5. PTOLEMY II Graphical Tool for simulation Design. Example of splitting the data generated by the Ramp module into two output devices

For the SimSWE project, the toolkit will be designed and implemented using the means of Matlab/Simulink tools. One of the main reasons of using Matlab-based tools is the ease and wide range of generating the random and probabilistic data. Besides, the Matlab syntax and paradigm is very widely known and used in the academia. The ease of changing the modules and recoding parts of the subroutines due to the scripted nature of it, makes it easy to adjust and re-tailor the relationships between the modules "on the fly".

6. Simulation modules

As mentioned in the previous section, the tool will be able to simulate both micro-processes of the development system, such as builds, tests coding etc., as well as the environment parameters such as development team expertise, budget etc.

The picture below demonstrates the simplistic format of a simulation module. The parameters of the environment are set within the Director, and the main module configuration parameters include the type of data generated (number, string, percentage, Boolean, array of values etc.), the ways the it is generated (triggered by the input, randomly or in timely manner), and the probability of each way.

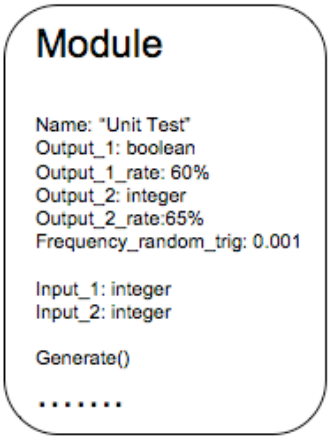

Figure 6 UnitTest module example.

Input can be the output from the previous module or modules, environment setting, or the self generated data.

For example, the commit event can be generated after the module of build return a successful result, or just generated after every interval of time (which can reflect a certain methodology or merely developer's habit). Sometimes evens can happen spontaneously without anything triggering them (random cost change, random build, etc). As shown on the picture, a simulation designer will define the probability of each way of triggering an event, as one of the parameters of the module. Each module can have several inputs that can be processed inside the module.

Output can be a number (number of build failures), Boolean value (result of the unit test), text (generated code), or a list of values (results of all unit tests). As it is implemented in other simulation tools, the module can have multiple outputs. For example, output of the unit test simulation module can return the result of each test (a list of Boolean 'pass/fail' values) as well as, the total test time, a number of procedures covered by the test, etc.

6.1 Examples of the modules

This section gives examples of the basic modules that will be present in the toolkit, including the input and output types and relationships to other modules. Obviously, some modules would use the data previously generated by them.

Code churn (CodeChurn)

Description: number of lines of code added, changed, or deleted

Input parameters:
- number_of_modules(number)
- LOC(number)

Settings:
- average_code_added(number)
- average_code_deleted(number),
- average_code_changed(number)

Output parameters:
- code_added (number)
- code_deleted(number)
- code_changed(number)

Data flows from (simulation modules):
- Modules
- TotalLoc

Builds (Builds)

Description: rate of builds including the results. The usual practice is to start building after the successful unit test, or update

Input parameters:
- code_churn(number)
- modules(number)
- LOC(number)

- previous_build_results(array of Boolean)

Settings:
- average_build_rate(number)
- average_failure_rate(number)

Output parameters:
- num_of_builds (number)
- build_results(array of Boolean)
- build_time (number)

Data flows from (simulation modules):
- UnitTest
- TotalLoc
- Modules
- CodeChurn
- Builds

Unit test (UntiTest)

Description: unit tests performed

Input parameters:
- number_of_modules(number)
- LOC(number)

Settings:
- average_test_coverage_per_unit (number)
- average_failure_rate (number)
- average_code_changed(number)

Output parameters:
- number_of_tests (number)
- test_coverage(number)
- test_results(array of Booleans)
- test_time(number)

Data flows from (simulation modules):
- Modules
- TotalLoc
- CodeChurn

System or project components (Modules)

Description: total number of components of the project. Can be objects, modules, procedures.

Input parameters:
- current_modules (number)

Settings:
- average_modules_added(number)

Output parameters:
- modules(number)

Data flows from (simulation modules):
- Modules

Total number of lines of code (TotalLoc)

Description: total number of lines of code in the project

Input parameters:
- number_of_modules(number)
- development_time(number)
- current_loc(number)

Settings:
- average_code_added(number)

Output parameters:
- loc(number)

Data flows from (simulation modules):
- Modules
- TotalLoc
- CodeChurn

Commits (Commits)

Description: commits to the repository

Input parameters:
- code_churn (number)

Settings:
- probability_random_commit(number)

Output parameters:
- loc(number)
- commit_results(array of Booleans)
- commit_time(number)

Data flows from (simulation modules):
- TotalLoc
- CodeChurn
- Build
- UnitTest

Issues (Issues)

Description: project issues, as they appear or get closed

Input parameters:
- code_churn (number)
- modules(number)
- current_issues(number)

Settings:
- average_issues_open(number)
- average_issues_closed(number)
- average_priority(number)

Output parameters:
- issues_open(array of strings (IDs) and numbers (priority levels))
- issues_closed(Array of strings (IDs))

Data flows from (simulation modules):
- TotalLoc
- CodeChurn
- Issues

Implemented requirements (ReqImplemented)

Description: client satisfaction as measured by the implemented requirements

Input parameters:
- current_reqs (number)

Settings:
- average_requirements(number)

Output parameters:
- requirements(number)

Data flows from (simulation modules):
- ReqImplemented

The list of these modules is incomplete, and the modules will be designed as the project is being developed. However, the terms in which the modules will be defined, are consisted with the given examples, and (more importantly) the GQM paradigm.

The laws of data genertion can be obtained from the specific

7. Applications of simulation

The most common question that arises regarding the use of simulation toolkit has to do with the simulation value. The argument is that the simulation has no value unless it reflects the real-life behaviors and trends. In other words, the outcome of the simulation is "pre-programmed" and hence, gives us no information, especially if the programmed behavior is based on the "theoretical" or random values. This argument is only valid, if we assume that the main use of the simulation is prediction of the real-life data. However, prediction is not the only and maybe one of the least uses of simulation toolkit.

The other applications of the toolkit described in this section are:

- Investigation of behavior and characteristics in controlled environment
- Investigation of macro behavior
- Understanding the impact and sensitivity of parameters
- Investigation of the methods, and processes under exact same conditions
- Design of more focused experiments
- Enhancement of causal analysis
- Test of the measurement tools

These points are described in the following sections

7.1 Investigation of behavior and characteristics in controlled environment

In order to discover a certain trend or find a relationship between the parameters, it is often required to look at the process from different viewpoints under different circumstances. For example, in order to claim that higher build rate leads to less build failures, we need to go through the process having several build rate settings (e.g. low, medium, high and extremely high), and look at the outcome. While running the process in real life under different circumstances might be very challenging, in the simulation, it will merely be achieved by altering the build rate parameters. After discovering the best parameters for a certain hypothesis support, they can be applied to a real-life project.

Running the projects with the multiple parameters, tuning them "on the go" can be a very challenging and costly task.

7.2 Investigation of macro behavior

Every developer, or every team has very unique characteristics that, though may vary from project to project, are, in general, consistent. In order to discover a certain trend under any given circumstances, it is required to look at several runs of the project to make sure that the behavior is as expected. The information given by the single run of the project with given parameters is not reliable enough, because as it varies within a certain range of outcomes, it can still give an extreme and highly improbable result. In order to eliminate "outliers" the project needs to be run multiple times under the same parameters. This allows us to investigate the average behavior, or average outcome.

In the example of requirements implementation strategy, for a given value of requirements volatility, sometime one strategy "wins over" the other, and sometimes the result is opposite. In order to gain confidence in what strategy is more effective (on average), the simulation is run multiple times, and the numbers of "wins" for each strategy are then compared.

7.3 Understanding the impact and sensitivity of parameters

When looking at the processes and projects, it is often important to know which parameters contribute into the observed change. Since the processes are usually quite complex, and multiple parameters would play different roles in the outputs, it can be crucial for a manager or designer to determine, which ones cause the change the most, and also, how sensitive the process is to the change of certain parameters. This can help organize the process in a way focusing on "important" parameters first, and "secondary" - after.

Using a simulation in this case would also be preferable, since the effect of every change in the parameter values can be almost immediately observed, while in real life, one will have to wait

until the end of the process, project or iteration to see the effect of the change.

7.4 Investigation of the methods and processes under exact same conditions

In order to observe a phenomenon, or confirm a hypothesis, it is important to see that the results of the "runs" are consistent. In other words, running the process under exactly the same conditions will (on average) result in exactly the same outcome.

Real-life projects, even related and/or when performed by the same developer or team of developers, can differ in a multitude of ways. Some of the differences may appear insignificant, but the accumulated effect can change the outcome greatly. This is why running project under exactly same settings is very challenging and even unrealistic.

The simulation, even when based on the random and probabilistic parameters, can be run as many times as needed with the values for probability or distribution remaining the same. When observing the results of such multiple runs, we can see whether the outcome is consistent (on average, even with possible "outliers").

7.5 Design of more focused experiments

Designing experiments that involve complicated and long-term processes such as software development process is very important for discovering trends and relationship between certain parameters and the outcome of the process, strategy or method. Having empirical data about the process can help properly design and plan it. Failure in applying a strategy or method can have a very high cost for the company, however, running an experiment for discovering the effect of the strategy or method can cost even more. In this case, running an experiment is useless.

The ability of the simulation to have adjustable parameters and give the "immediate" results, as mentioned above, can show us which parameters are more influential, and how sensitive the process is to certain settings. This can help tremendously to design a software development process more precisely, gathering the data or paying a closer attention only to the parameters that, according to the simulation, have greater effect on the result. This reduces the effort of gathering, storing and analyzing the data, and makes the experiment more focused and efficient.

7.6 Enhancement of casual analysis

When the project is run, very often we want to focus not as much as the final outcome, but rather on the dynamics of changes, as they occur, and how they are related to each other. Knowing the casual connection between parameters can help us to apply the strategies "as we go".

For example, if the current (or immediate) goal of the project is to minimize the faulty executions of the program (as opposed to the "global" one which is, for example, to minimize the cost, finish ahead of the schedule, etc.), one solution could be to maximize the testing rate. This may cause in longer project time(for a given iteration), and might not fulfill the ultimate goal, but will satisfy the "immediate" one – maximizing the requirement satisfaction.

In this example, we claim to possess the knowledge on the casual relationships between parameters and settings, as opposed to the relationships between the parameters and final outcome. As discussed above, this kind of knowledge can only be trusted, when obtained from multiple projects, teams, strategies and runs. The simulation can help us achieve this, without spending as much resources as the real-life experiments would.

7.7 Test of measurement tools

The software development measurement tools are the systems that perform an analysis of various dimensions of the project information. These data can be gathered both automatically, and manually. The analysis can include the prediction of the outcome (cost, time, user satisfaction etc.), indication of the project's "health" (failure rate is within a reasonable range, cost variation is not crucial etc.), demonstrating of certain trends (test-driven development, agile vs. traditions etc.) or other analysis.

For the measurement system to be reliable, it needs to be tested for a wide range of situations and parameters. For example, the constructed "health" analyzer has to always return the positive outcome for "healthy" projects (however we define "health") and negative outcome for "unhealthy" ones. "Feeding" the analyzer with the real-life data can be very time consuming, since the data needs to be collected over a vast period of time. Being able to

simulate the data can be extremely helpful in situation like this, because, as mentioned, it does not matter how we define a "healthy" project, since the parameters of the simulation can be adjusted to any definition of a "healthy" or "unhealthy" project.

7.8 Other notes on simulation use

One may argue that the parameters, laws and relationships obtained during the simulation will never be the same as the real ones. Again, such perception of the simulation usage is limited. As long as the simulation demonstrates the behavior "similar" to that of the real-life (even, in simplistic way, for example, showing a strong positive correlation, without giving an exact actual coefficients), it has a great value for the project management.

Another important note on the simulation use is that the simulated data and the analysis of these data need to be seen separately. As in the example with requirements prioritization strategy shows, the data is generated (simulated) independently from making the decision about it (strategizing). The simulation tool does not necessarily claim to find the best strategy or the method, but rather it gives the means to test the discovered or hypothesized methods or trends on a wide range of data. The laws of data simulation should not affect the decision made bout these data.

Also, simulation toolkit can help us run the project for the "entire" range of the values (no matter how you define the range of certain parameters), and analyze the data. If the outcome is statistically consistent for "all possible" parameters, it gives us confidence that the result also applies to the real-life projects, since the actual real-life parameters are most probably within the performed exhaustive runs.

8. Related work

One of the most interesting applications of the simulation system, was found in the work describing the ProSim/RA framework – a risk assessment system - which combines software process simulation with stochastic simulation. It consists of the following steps.

STEP 1: Define risk factors. Risk factors are attributes of project entities that are supposed to cause losses of a certain amount with a certain probability.

STEP 2: Define impact factors. Impact factors are attributes of projects that are supposed to be affected by variations of risk factors.

STEP 3: Define variation of risk factors. This includes the construction of a distribution function describing the probability of assuming a particular value.

STEP 4: Conduct sensitivity analyses.

STEP 5: Analyze simulation results. By this, rankings can be constructed that may serve as an input for risk prioritization. Typically, potential losses are associated with late product delivery (contract penalty), lacking product quality (rework cost), effort overrun (personnel cost).

This, seemingly simple and intuitive approach provides a very good systemstic process for risk analysis, as shown in the example presented in the paper. In this example, in order to test the suggested risk assessment framework, a simulated data from GENSIM (GENeric SSIMulator) is used. GENSIM is a generic prototype representing a waterfall-like software development process. The GENSIM model simulates the software development process from the end of the requirement analysis step through to the end of system testing.

Among several advatnages of using of simulation for the risk assessment, the most relevant is the one described in the previous sections: high, medium, and low business-priority defects – the task that can hardly be implemented in the real-life projects.

9. References

[1] Pfahl D., "System Dynamics and Goal Oriented measurement: a hybrid approach"

[2] Boehm, B W "Software Engineering Economics." Prentice Hall PTR, 1981

[3] D. Pfahl, K. Lebsanft "Using simulation to analyze the impact of software requirement volatility on project performance"

[4] Nagappan N, Ball T. "Explaining Failures Using Software Dependences and Churn Metrics"

[5] Pfahl D (2005) ProSim/RA – Software Process Simulation in Support of Risk Assessment

[6] Port D., Olkov A Using Simulation to Investigate Requirements Prioritization Strategies

[7] http://code.google.com/p/hackystat/wiki/ContinuousIntegrationICU

[8] Lee A., Neuendorffer C, Tutorial: Building Ptolemy II Models Graphically. Technical Report No. UCB/EECS-2007-129

[9] Noehm B. W. «Software Metrics. A Rigorous & Practical Approah» PWS Publishing Company, 1997

[10] Biffl S, Aurum A. et al «Value-Based Software Engineering», Springer 2006