A Probabilistic Model for Software Projects

Frank Padberg [⋆]

Fachbereich Informatik, Universität Saarbrücken, Germany

Abstract. A probabilistic model for software development projects is constructed. The model can be applied to compute an estimate for the development time of a project. The chances of succeeding with a given amount of time and the risk of deviating from the estimate can be computed as well. Examples show that the model behaves as expected when the input data are changed.

1 Introduction

At an early stage of a software development project the managers need to establish an estimate for the development time of the project. This is a difficult task since an estimate depends on many factors which are unknown at that time. The managers also need to analyze the risk of a project. They must find answers to more detailed questions such as

> What are the chances that the project will be successfully completed within two years?

The answer much depends on which *course* the project will take. Think of the course of a project as describing "what happens at what time". The course is not known in advance, so uncertainty and risk are inherent to any project. The best one can do is to make some sort of assessment based on one's *experience* with past projects. The central idea of this paper is to put such assessments on a solid mathematical basis by constructing a *probabilistic model* for software development projects. A first model is presented in this paper.

The core of the model consists of formulas for computing the *probabilities of the courses* that a project could take. A project is modelled as a random process whose *state* changes from time to time. This leads to a formal description of a course of the project as a sequence of states, see subsection 3.3. The transitions between the states are controlled by the *transition probabilities* from which the probabilities of the project's courses can be computed, see subsection 3.4.

The input data required for computing the transition probabilities are *statistical data* and *design data*. The statistical data are a measure for the pace progress was made with in previous projects, bringing in the experience made in the past.

[⋆] supported by a doctoral fellowship of the Deutsche Forschungsgemeinschaft DFG at the Graduiertenkolleg Informatik, Saarbrücken

The raw data needed to build up a database could be collected by the manager during running projects with little effort, see subsection 4.3. The design data are a measure for the degree of coupling between the software product's components, see subsection 4.6. The stronger the degree of coupling is the more likely it is that changes in one component will propagate into the other components. The design data have to be extracted from some early high-level design of the software.

Once the probabilities of a project's courses are known, the *chances of succeeding* with a given amount of time can be computed. This is done by summing up the probabilities of those courses that would take at most the given time to complete. An *estimate for the development time* is computed as the "weighted average" of the lengths of the courses, the weights being the probabilities of the courses. The probability or *risk* that the time actually needed to complete the project will exceed the estimate by some chosen amount of time can be computed, too. This provides an *error bound* for the estimate. See subsection 3.5.

The approach is not tailored to a specific software process model, see section 2. To get through the complicated mathematics, in the model presented here several simplifying assumptions were made:

- The number of development teams equals the number of components in the high-level design.
- The teams start working at the same time and keep working until their components are completed.
- The customer requirements do not change during the project.
- The number of components, the degree of inter-component coupling, and the complexity of each of the components remain the same through all changes of the high-level design.

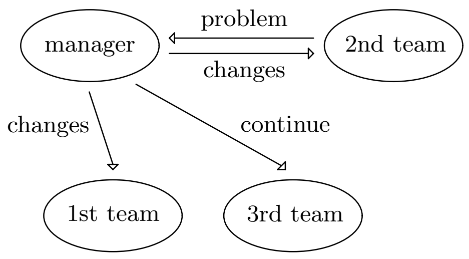
The assumptions are preliminary but limit the applicability of this particular model. In particular, volatility in the customer requirements is considered to be one of the leading causes for schedule and cost overrun, see [6]. Therefore, the model should be regarded as just a first step towards a comprehensive probabilistic model for software projects. It has not been tested using real world data so far, and it is not meant to be used in practice at this early stage.

There is much theoretical and empirical research on software project estimation. For an overview of the common models and many references see [10][11]. The common models, in particular the curve-fitting approaches, tend to produce unreliable estimates, see [3][4][5][6]. This might be explained by the fact that the models do not explicitly consider the uncertainty inherent to a project. There also are problems with calibrating the models and, most notably, with obtaining a valid size estimate as their basic input variable, see [1][4][7][8]. Since there is a strong need in the software industry for reliable estimates, increasing effort is being spent on new approaches such as machine learning, analogy and neural networks, see [12][13][14]. For an overview see [2].

The approach presented here differs from the existing models in several respects. First, it is based directly on modelling the courses of a software project. It emphasizes that the progress of a team depends on the other teams' progress. Second, it uses a standard mathematical framework for modelling a process whose line of development is not known in advance. This includes explicit modelling of the risk of a project. As far as I know, computing the chances of a project succeeding and computing error bounds for the estimate is not possible with the existing models. Third, the experience made with past projects enters in a standard and comprehensible way as statistical data which will automatically adjust to the local environment.

2 Software Projects

A software project is considered to consist of several development *teams* and a project *manager*. Based on some early high-level design, the software product gets divided into *components*. Each team is assigned one component and vice versa. The teams start working at the same time. There are no assumptions made about the software engineering methods or process models used by the individual teams. The teams work simultaneously, but not independently. They do not communicate directly, but there will be an interaction between them in case of a team detecting a *problem* with the system's design. Since the components are coupled, such a problem is likely to affect other teams as well, so the team reports the problem to the manager. The manager makes sure that the system's design gets revised to eliminate the problem. The team that has reported the problem waits until redesigning is finished. The other teams keep working. If there are additional problem reports while the design is being revised, they are taken into account, too. When the *redesign* is completed, the manager tells each team whether it is affected by the design changes or not.



In this example, the redesign results in changes to the components that the first and second team are working on. The third team just continues without changes. It may happen that a team that has already finished has to go over its component again because of design changes. To sum up, the progress of a team depends on the other teams' progress, the teams being linked through the system's design. When all teams have reported to the manager that they have finished developing, the components are put together and the system gets tested. If errors are detected, a new *development cycle* begins.

The model will describe a development cycle probabilistically.

3 The Model

3.1 Time

In the model, time is discrete. The time axis gets subdivided into periods of equal length, called *time slices*. The length of a slice can be chosen as is suitable. Think of a slice as corresponding to, say, a month in real time. Whether a team has detected a problem or has finished working during a slice will be determined at the end of the slice. The exact point of real time at which something happened will be disregarded.

There is another subdivision of the time axis. As a project advances, the system's design will be revised from time to time because of problems. The time span between two consecutive redesigns is called a *phase*. Each phase consists of a number of slices that may vary from phase to phase.

3.2 States

At any point of discrete time the *state of a team* is represented by a natural number, zero included. This number is to express the progress that the team has made up to this point. One could think of several ways of measuring a team's progress. In this version of the model the state of a team is defined as the number of slices that have passed since the team last reported a problem or was last interrupted by the manager because of being affected by changes in the system's design. A value of infinity indicates that the team has finished developing.

The *state of a project* at the end of a phase is defined as a vector

$$\zeta = (\zeta_1, \zeta_2, \ldots \zeta_N)$$

where ζ_i is the state of team number i at the end of the phase and N is the number of teams. A state vector of $\underline{\infty} = (\infty, \infty, \ldots \infty)$ means that all the teams have finished. Note that $(4, \infty, 2)$ would not be a valid state vector, because at the end of any phase either all teams have finished or at least one of the teams is set back to zero.

3.3 Course of a Project

The *course of a project* is modelled as a sequence

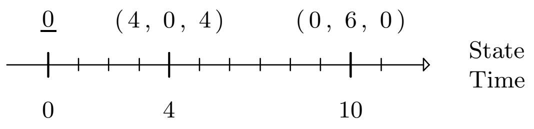
$$\zeta(1), \zeta(2), \ldots$$

of states, where $\zeta(j)$ is the project's state vector at the end of phase j. It is assumed that the number of teams doesn't change during the project. The sequence of states has to be supplemented by the sequence of numbers

$$d_1, d_2, \ldots$$

giving the length of each of the project's phases.

For example, this might be the first two phases of a three-team project:



State vector $\underline{0} = (0, 0, 0)$ corresponds to the initial high-level design. The project has advanced from state $\underline{0}$ after 4 slices (first redesign) into state $(4, 0, 4)$ and after 6 more slices (second redesign) into state $(0, 6, 0)$.

3.4 Transition Probabilities

To expand the formal description of a project's courses into a probabilistic model, define the *transition probability*

$$\mathrm{P}_{\zeta}(d, \eta)$$

to be the probability for ending the next phase after exactly d slices with state η given that the previous phase ended with state ζ. The formulas for the transition probabilities will be constructed step-by-step in section 4.

Note that the transition probabilities do not depend on any information about a project's history except its current state ζ. In particular, the transition probabilities do not depend on the particular sequence of states that the project passed through before reaching the current state. The model therefore will be a *Markov process*. For such a modelling to make sense the state must be defined to contain all relevant information about the project's past.

When all the transition probabilities are known one can compute the probability $\mathrm{P}(\omega)$ that a project will take a particular course ω simply by multiplying the corresponding transition probabilities,

$$\begin{aligned} \mathrm{P}(\omega) \;=\; & \mathrm{P}_{\underline{0}}\big(d_1, \zeta(1)\big) \cdot \mathrm{P}_{\zeta(1)}\big(d_2, \zeta(2)\big) \cdot \\ & \cdot \;\ldots\; \cdot \mathrm{P}_{\zeta(k-1)}\big(d_k, \zeta(k)\big). \end{aligned}$$

For example, the probability that a project will advance in its first two phases like in the figure above is $\mathrm{P}_{\underline{0}}\big(4, (4, 0, 4)\big) \cdot \mathrm{P}_{(4,0,4)}\big(6, (0, 6, 0)\big)$.

The space Ω describing all possible courses of a project contains arbitrarily long sequences ω of pairs (d, η) corresponding to a successful outcome, but also infinite sequences corresponding to a never-ending project. This space is uncountable, so it can't be treated within the framework of discrete probability theory. It is well-known how to formally construct from the transition probabilities a non-discrete probability measure on that space using the methods of general measure theory. For the purpose of this paper one can avoid the technical difficulties as follows. For each project there naturally exists a "final deadline" of, say, x_0 slices. If this deadline is exceeded, the project will be cancelled as a failure. Only the probabilities of those courses get considered which have completed before this deadline is reached, together with the remaining probability that the project will exceed the final deadline.

3.5 Estimates

The model can be applied as follows. Define Ω^+ to consist of those finite sequences ω of pairs $(d_j, \zeta(j))$ which correspond to a successful outcome of the project, that is, for which just the last state equals infinity. Denote by $|\omega|$ the number of phases in ω. Define a function

$$f : \Omega^+ \longrightarrow \mathbb{N}$$

which assigns to each finite course ω its *length of time*

$$f(\omega) = \sum_{j=1}^{|\omega|} d_j$$

by summing up the lengths of its phases. The probability that the project will take exactly x slices to complete is equal to

$$\varphi(x) = \sum_{\{\omega : f(\omega) = x\}} \mathrm{P}(\omega) .$$

The sum runs over all successful courses ω whose length of time is x slices. The probabilities $\mathrm{P}(\omega)$ get computed from the transition probabilities, as is described in the preceding subsection. Observations that will be made in subsection 4.3 ensure that the number of different states of a project is finite and that the length of a phase is bounded. It follows that the set $\{\omega : f(\omega) = x\}$ is finite, and the sum has only finitely many summands.

The *chances of succeeding* with the project before the final deadline of x_0 slices is exceeded are equal to the probability

$$\Phi(x_0) = \sum_{x \leq x_0} \varphi(x) .$$

The more development time there is at the beginning of the project the greater are the chances of succeeding, because Φ is an increasing function. Suppose that a project is approaching the final deadline. If the project's state is $\underline{0}$ after y slices, the chances of still succeeding in the remaining time are only equal to $\Phi(x_0 - y)$ in the model. The closer the project is to the deadline, the smaller these chances of still succeeding are.

As an *estimate for the development time* of a project compute the number

$$\mathrm{E}_{time} = \sum_{x \leq x_0} x \cdot \varphi(x) + x_0 \cdot (1 - \Phi(x_0)) .$$

This is a "weighted average" of the lengths of the courses cut off at the final deadline x_0, the weights being the probabilities of the courses, because

$$\sum_{x \leq x_0} x \cdot \varphi(x) = \sum_{\{\omega : f(\omega) \leq x_0\}} f(\omega) \cdot \mathrm{P}(\omega) .$$

Note that E_{time} in general is not equal to the expected value of the function f expanded to a random variable on the space Ω mentioned in the preceding subsection. It can be shown that E_{time} converges from below to the expected value as x_0 increases, provided that the expected value exists.

To obtain *error bounds,* compute the probability that the length of a project will exceed the estimate by at most some number c of slices as

$$\mathrm{P}(0 < f - \mathrm{E}_{time} \leq c) = \Phi(\lfloor \mathrm{E}_{time} + c \rfloor) - \Phi(\lfloor \mathrm{E}_{time} \rfloor) .$$

All the formulas can be pictured using a bar chart for the probabilities $\varphi(x)$. An example is given in subsection 5.2.

4 Probabilities

4.1 Approach

The formulas for the transition probabilities are constructed in two major steps. First, consider the probabilities of the events that can happen during a phase. Second, consider the probabilities for the effects that the redesign at the end of the phase can have on the teams. More precisely, set

$$\mathrm{P}_\zeta(d, \eta) = \sum_{v : d} \mathrm{P}_\zeta(v) \cdot \mathrm{P}_\zeta(\eta \mid v) .$$

Here, $\mathrm{P}_\zeta(v)$ is defined to be the probability that the phase develops according to v, which denotes a course of the phase. The possible courses will be formalized in the next subsection. $\mathrm{P}_\zeta(\eta \mid v)$ is defined to be the conditional probability that the phase ends with state η assuming that it developed according to v. The sum runs over all possible courses v of length exactly d slices. It corresponds to the fact that different courses of a phase may lead to the same next state. As is indicated by the lower index ζ, all probabilities depend on the previous state of the project, that is, the state at the beginning of the phase.

4.2 Course of a Phase

The course v of a phase is known when its length d and the numbers

$$1 \leq k_1 < \ldots < k_n < d \qquad \text{and} \qquad 1 \leq l_1 < \ldots < l_m \leq d$$

are specified. The numbers k_j are the different points in time at which problems get reported, measured as the number of slices that have passed since the beginning of the phase. The numbers l_j are the different points in time at which teams report that they have finished. Define the set K_j to contain the numbers of the teams that report a problem at time k_j. The set L_j corresponds to time l_j. In addition, define the set L_0 to consist of the numbers of those teams which do not report anything during the phase. This includes the teams which already are in state infinity since an earlier phase.

4.3 Statistical Data

To compute the transition probabilities, some input data are required about the pace progress was made with in past projects. Define the *base probabilities*

$$\mathrm{P}(\mathrm{E}_k^i) \quad \text{and} \quad \mathrm{P}(\mathrm{D}_k^i)$$

to be the probabilities that team number i will report a problem (event E_k^i) or will reach state infinity (event D_k^i) exactly k slices after having been interrupted for the last time by the manager because of changes in the system's design. Since the D_k^i and E_k^i are disjoint events, for fixed i

$$\sum_k \mathrm{P}(\mathrm{E}_k^i) + \mathrm{P}(\mathrm{D}_k^i) = 1 .$$

A team will eventually reach state infinity or report a problem if it doesn't get interrupted. This gives a bound k_0 such that $\mathrm{P}(\mathrm{D}_k^i) = \mathrm{P}(\mathrm{E}_k^i) = 0$ for all $k > k_0$, so the number of states of a project is finite.

The base probabilities have to be computed from raw data about the courses of past projects. During a running project, the raw data needed for future use get collected like this. Given that a slice corresponds to one month, the manager has to write down at the end of each month

- the number of each team which reported a problem;
- the number of each team which has finished;
- whether there was a redesign;
- if there was a redesign, the number of each team which was affected by design changes.

The base probabilities get computed as average values on a team-by-team basis from the raw data sets of as many projects as possible. If the data sets come from a single organization, the values will automatically adjust to the specific environment in that organization. Updating the database and computing the base probabilities could be supported by a tool.

For example, suppose that we have the data sets from 15 projects and that a particular team participated in all the projects. Assume that the team has been interrupted by a redesign for a total number of 35 times during these projects (this is 2.3 times on average). Also assume that it happened 4 times that the team reported a problem exactly 2 months after it had been interrupted by a redesign for the last time or right after the project had started. The base probability $\mathrm{P}(\mathrm{E}_2)$ for the team would be set to $4/(35+15)$, or 8 percent.

The base probabilities for a team depend upon various factors, for example

- the software process employed by the team;
- the complexity of the component that the team has to develop;
- the skills and the experience of the team members.

Therefore, the manager should distinguish between different team productivity levels and component complexity classes when computing the average values, if the database is sufficiently large and detailed.

In addition to the base probabilities, define the *probability of redesign time*

$$\gamma(k)$$

to be the probability that it will take exactly k slices to redesign the system if a problem was reported, provided that there are no further problem reports in the meantime. The probabilities of the redesign times get computed from the raw data sets of past projects just like the base probabilities. The values depend on the complexity of the software product's design. There certainly is a bound on the number of slices that a redesign will take. Along with the bound mentioned above, the length of a phase is bounded. For later use, define

$$\Gamma(k) = \sum_{s=0}^{k} \gamma(s).$$

4.4 Advance of a Team

Suppose that team number i is in state $\zeta_i \neq \infty$ at the beginning of some phase. The probabilities describing how the team will advance during the phase can be computed using the team's base probabilities. For example, the probability that the team will report a problem after k slices is equal to the conditional probability

$$\mathrm{P}(\mathrm{E}^i_{\zeta_i+k} \mid \mathrm{B}^i_{\zeta_i}).$$

Conditioning by the event

$$\mathrm{B}^i_{\zeta_i} = \overline{\bigcup_{s=1}^{\zeta_i} \left(\mathrm{E}^i_s \cup \mathrm{D}^i_s \right)}$$

takes into account the knowledge that the team already has been working prior to the current phase for ζ_i slices without having been interrupted. It follows from $\mathrm{E}^i_{\zeta_i+k} \subset \mathrm{B}^i_{\zeta_i}$ that

$$\mathrm{P}(\mathrm{E}^i_{\zeta_i+k} \mid \mathrm{B}^i_{\zeta_i}) = \frac{\mathrm{P}(\mathrm{E}^i_{\zeta_i+k})}{\mathrm{P}(\mathrm{B}^i_{\zeta_i})}.$$

Similarly, the probability that the team will have finished after l slices equals

$$\mathrm{P}(\mathrm{D}^i_{\zeta_i+l} \mid \mathrm{B}^i_{\zeta_i}).$$

The probability that the team will not report anything for a period of d slices is equal to

$$\mathrm{P}(\mathrm{B}^i_{\zeta_i+d} \mid \mathrm{B}^i_{\zeta_i}).$$

As can be seen from the definition, the probability of event $\mathrm{B}^i_{\zeta_i}$ is given by

$$\mathrm{P}(\mathrm{B}^i_{\zeta_i}) = 1 - \sum_{s=1}^{\zeta_i} \Big(\mathrm{P}(\mathrm{E}^i_s) + \mathrm{P}(\mathrm{D}^i_s) \Big).$$

If $\zeta_i \neq \infty$ but $\mathrm{P}(\mathrm{B}^i_{\zeta_i}) = 0$, the conditional probabilities involving $\mathrm{B}^i_{\zeta_i}$ are set to zero. If $\zeta_i = \infty$, they are set to one.

4.5 Probability $\mathrm{P}_\zeta(v)$

As a first step, the probabilities that the individual teams will advance according to the given course v are multiplied. For example, if the second team reports a problem at time k_j this adds the factor

$$\mathrm{P}(\mathrm{E}^2_{\zeta_2+k_j} \mid \mathrm{B}^2_{\zeta_2}).$$

Multiplying the individual probabilities is appropriate because the problems reported by one or more teams during a phase will have no effect on the other teams until the end of that phase. By that time the new design is completed and the manager will interrupt the teams which are affected by changes. Therefore, the teams work independently of one another during the phase.

As a second step, a factor of

$$1 - \Gamma(k_{j+1} - k_j - 1)$$

has to be added for each $j = 1, \ldots n-1$. This corresponds to the fact that redesigning didn't get accomplished in the period between the problem reports at time k_j and those at time k_{j+1}. Since the new design eventually is completed $d - k_n$ slices after the problem reports at time k_n, this adds another factor

$$\gamma(d - k_n).$$

As a third step, since both v and ζ were arbitrarily chosen, probability $\mathrm{P}_\zeta(v)$ has to be set to zero if it is not possible for a phase to develop according to v when starting from ζ. If v contains problem reports, the only condition is that teams which already are finished since an earlier phase won't report anything,

$$\zeta_i = \infty \implies i \in L_0 .$$

Define

$$\mathrm{I}_\zeta^{(0)}(v)$$

to be one if v fulfils that condition and to be zero if not. If v does not contain problem reports, the phase ends if and only if all teams that still have been working at the beginning of the phase eventually reach state infinity, the last one after d slices,

$$l_m = d \quad \text{and} \quad \bigcup_{j=1}^{m} L_j = \{ i \mid \zeta_i \neq \infty \} .$$

Define

$$\mathrm{I}_\zeta^{(2)}(v)$$

to be one if v fulfils that (stronger) condition and to be zero if not.

Summing up, the formula for probability $\mathrm{P}_\zeta(v)$ looks like this if there are problem reports during the phase:

$$\begin{aligned}
\mathrm{P}_\zeta(v) \;=\; & \prod_{j=1}^{n} \prod_{i \in K_j} \mathrm{P}(\mathrm{E}^i_{\zeta_i + k_j} \mid \mathrm{B}^i_{\zeta_i}) \\
\times & \prod_{j=1}^{m} \prod_{i \in L_j} \mathrm{P}(\mathrm{D}^i_{\zeta_i + l_j} \mid \mathrm{B}^i_{\zeta_i}) \\
\times & \prod_{i \in L_0} \mathrm{P}(\mathrm{B}^i_{\zeta_i + d} \mid \mathrm{B}^i_{\zeta_i}) \\
\times & \prod_{j=1}^{n-1} (1 - \Gamma(k_{j+1} - k_j - 1)) \\
\times & \; \gamma(d - k_n) \quad \times \quad \mathrm{I}_\zeta^{(0)}(v) .
\end{aligned}$$

If there are no problem reports during the phase, the formula looks like this:

$$\begin{aligned}
\mathrm{P}_\zeta(v) \;=\; & \prod_{j=1}^{m} \prod_{i \in L_j} \mathrm{P}(\mathrm{D}^i_{\zeta_i + l_j} \mid \mathrm{B}^i_{\zeta_i}) \\
\times & \; \mathrm{I}_\zeta^{(2)}(v) .
\end{aligned}$$

4.6 Design Data

To compute the transition probabilities, some input data about the degree of coupling between the software product's components are required. For nonempty subsets K and X of the set of components $\{1, \ldots N\}$ the *dependency degree*

$$\alpha\left(K, X\right)$$

is defined to be the probability that changes in the system's design will extend over exactly the components X given that the problems causing the redesign were detected in the components K. A component gets identified with its number here to simplify notation. For fixed K,

$$\sum_{X} \alpha\left(K, X\right) = 1$$

because design changes will extend over one and only one of the sets X. Since a problem that was detected in a particular component will result in changes to that component, it is required that

$$K \not\subseteq X \implies \alpha\left(K, X\right) = 0 .$$

The dependency degrees have to be extracted from the high-level design of the software product being developed. They can be viewed as a partial measure for the complexity of a high-level design.

4.7 Probability $\mathbf{P}_{\zeta}\left(\eta \mid v\right)$

For this subsection, a state vector $\zeta \neq \underline{\infty}$ and a course v of a phase are fixed. It suffices to consider only such pairs (ζ, v) for which $\mathrm{I}_{\zeta}^{(0)}(v) = 1$, respectively, $\mathrm{I}_{\zeta}^{(2)}(v) = 1$, see the third step in subsection 4.5.

Given ζ and v, not every state η is a valid next state for the project. For example, teams that were in state infinity at the beginning of the phase and are not affected by design changes will still be in state infinity at the beginning of the next phase. Therefore, probability $\mathrm{P}_{\zeta}(\eta \mid v)$ has to be set to zero if η is not a valid next state.

Suppose that according to v there are problem reports during the phase. Denote by X the set containing the numbers of the teams which are affected by design changes. To characterize the valid next states, note that the next state already is uniquely determined when X is specified. This can be seen as follows.

- From the definition for the state of a team it follows that the next state of a team which is affected by changes is zero :

$$i \in X \implies \eta_i = 0 .$$

- If a team has been working during the whole phase and is not affected by changes it will have advanced by d slices:

$$i \in L_0 \setminus X \text{ and } \zeta_i < \infty \implies \eta_i = \zeta_i + d\,.$$

- If a team has been in state infinity and is not affected by changes it will remain in that state:

$$i \in L_0 \setminus X \text{ and } \zeta_i = \infty \implies \eta_i = \infty\,.$$

- If a team has finished working during the phase and is not affected by changes it will be in state infinity:

$$i \in L_j \setminus X \implies \eta_i = \infty\,.$$

In addition, it is required that

$$K = \bigcup_{j=1}^{n} K_j \subset X$$

since a team which has reported a problem will be affected by changes. On the other hand, since only those teams which are affected by changes will have zero as their next state, the set X is determined if η is given,

$$X = \{ i \mid \eta_i = 0 \}\,.$$

Therefore, a state vector η is a valid next state of the project if and only if the values of its nonzero entries are as described above and $K \subset \{ i \mid \eta_i = 0 \}$. Define

$$\mathrm{I}^{(1)}_{\zeta,v}(\eta)$$

to be one if η is a valid next state and to be zero if not.

For a valid next state, the conditional probability $\mathrm{P}_\zeta(\eta \mid v)$ is equal to the probability that just the teams with numbers from $X = \{ i \mid \eta_i = 0 \}$ will be affected by changes. This probability equals the dependency degree

$$\alpha\big(K,\ \{ i \mid \eta_i = 0 \}\big)\,.$$

It implicitly is assumed here that the dependency degrees remain about the same through all changes of the system's design. If K is nonempty, the formula for probability $\mathrm{P}_\zeta(\eta \mid v)$ thus is given by

$$\begin{aligned}\mathrm{P}_\zeta(\eta \mid v) &= \alpha\big(K,\ \{ i \mid \eta_i = 0 \}\big)\\ &\quad\times\ \mathrm{I}^{(1)}_{\zeta,v}(\eta)\,.\end{aligned}$$

It remains to consider the case that according to v all teams have finished by the end of the phase. The only possible next state is $\eta = \underline{\infty}$. If K is empty, the formula for probability $\mathrm{P}_\zeta(\eta \mid v)$ thus is

$$\mathrm{P}_\zeta(\eta \mid v) \;=\; \begin{cases} 1 & \text{if } \eta = \underline{\infty} \\ 0 & \text{otherwise}\,. \end{cases}$$

4.8 Proofs

The formulas for the transition probabilities $\mathrm{P}_\zeta(d,\, \eta)$ yield a Markov process model only if for each state $\zeta \neq \underline{\infty}$ the probabilities for the transitions from state ζ to some other state sum up to one. That is,

$$\sum_{d,\,\eta} \mathrm{P}_\zeta(d,\, \eta) \;=\; 1\,.$$

The proof of this is involved and is given in [9].

5 Examples

Some small examples are to show that the model behaves as expected when the input data are changed. Due to their large number, the computed transition probabilities are not listed. A table containing the numbers for all the charts printed below is available by email from the author. Recall that the following input data have to be specified:

- the number of teams N;
- the base probabilities $\mathrm{P}(\mathrm{D}_k^i)$ and $\mathrm{P}(\mathrm{E}_k^i)$ for each team;
- the probabilities of redesign times $\gamma(k)$;
- the dependency degrees $\alpha\big(K,\, X\big)$.

All probabilities are specified as percentages.

5.1 Input Data

There are three teams. A slice in discrete time corresponds to a month in real time. Each of the teams will have finished or will report a problem after at most six months, provided that it doesn't get interrupted in the meantime. The base probabilities for the teams are:

k	D_k^1	E_k^1	D_k^2	E_k^2	D_k^3	E_k^3
1	0.0	10.0	10.0	30.0	10.0	5.0
2	9.0	9.0	18.0	12.0	8.5	8.5
3	21.6	7.2	15.0	6.0	20.4	20.4
4	21.6	4.3	3.6	3.6	15.0	2.7
5	12.1	1.7	0.5	1.1	7.1	1.0
6	3.1	0.4	0.1	0.1	1.3	0.1

Redesigns will be completed within a month,

$$\gamma(1) = \texttt{100}.$$

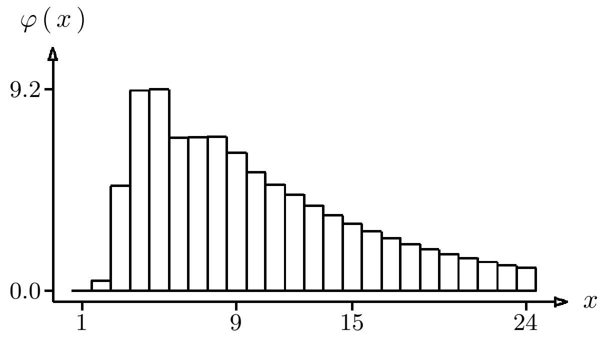
The values for the dependency degrees are given in the next table. Several entries are zero since this is required if $K \not\subseteq X$, see subsection 4.6. The non-zero entries are assumed to be uniformly distributed for each K in this example.

$\alpha\big(K, X\big)$	1	2	3	1, 2	1, 3	2, 3	1, 2, 3
1	25	0	0	25	25	0	25
2	0	25	0	25	0	25	25
3	0	0	25	0	25	25	25
1, 2	0	0	0	50	0	0	50
1, 3	0	0	0	0	50	0	50
2, 3	0	0	0	0	0	50	50
1, 2, 3	0	0	0	0	0	0	100

The table contains one line for each K and one column for each X. The set braces are left out.

5.2 Results

Suppose that a project has last subsection's input data. The following chart shows up to $x = 24$ the values of the probabilities $\varphi(x)$ that the project will take exactly x months to complete.



The peak value is $9.2\,\%$. The first value is zero since the probability $\mathrm{P}(\mathrm{D}_1^1)$ that the first team will have finished after one month is zero. The chances of succeeding within two years correspond to the sum of the areas of the bars, which is about $92\,\%$ for this project. This answers for the sample project the question posed in the introduction. The $50\,\%$ threshold is reached after 9 months, the $75\,\%$ threshold after 15 months.

Choose a final deadline of $x_0 = 24$ months. The estimate for the development time then is

$$\mathrm{E}_{time} = 11.1$$

months. To picture this using the chart, multiply each bar x with its height $\varphi(x)$ and sum up. Then add a small adjustment which takes into account the remaining probability for exceeding 24 months of development time.

The standard deviation for the chart is 6.5 months. The probability that the development time actually needed to complete the project does exceed the estimate, the delay being at most the standard deviation, is equal to

$$\Phi(17) - \Phi(11) = 20\,\%.$$

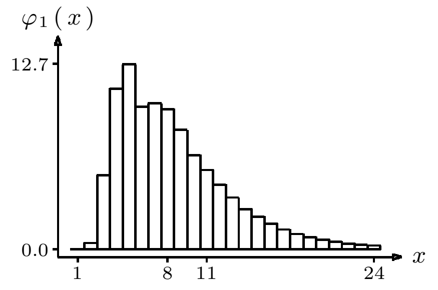
This error bound corresponds to the sum of the areas of the bars numbered 12 through 17 in the chart.

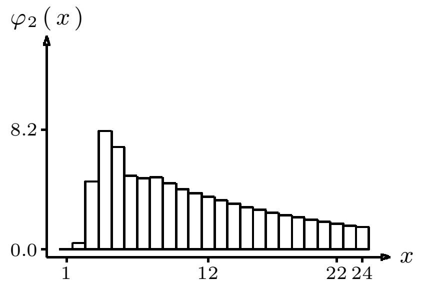
5.3 Different Dependency Degrees

To illustrate the influence of the values of the dependency degrees on the results, consider the "best case" and the "worst case" given by

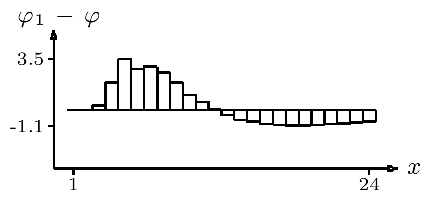
$$\alpha\big(K,\ K\big) = \texttt{100} \qquad \text{and} \qquad \alpha\big(K,\ \{1, 2, 3\}\big) = \texttt{100}.$$

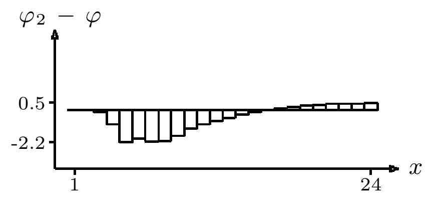
The best case means that no teams will be affected by changes in the system's design other than those which reported a problem. The worst case means that all teams will be affected by changes no matter which teams reported a problem. The remaining input data are fixed. The charts for the resulting functions φ_1 and φ_2 look like this:





For the best case, the chances of succeeding within two years have improved to 99 %. The 50 % threshold is reached after 8 months, the 75 % threshold after 11 months. For the worst case, the chances of succeeding within two years are only 79 %. The 50 % threshold is reached not until 12 months, the 75 % threshold not until 22 months. The following charts compare the values to the initial example:

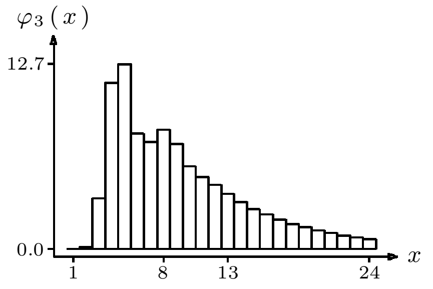


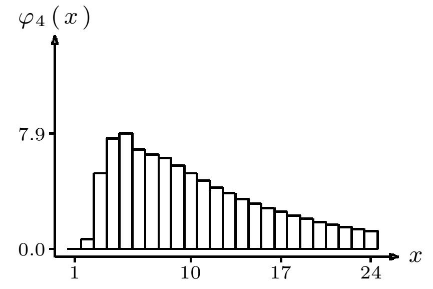


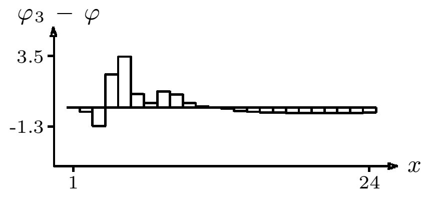
The differences show that the first bars for the best case lie above the corresponding bars of the initial example. There is a turning point because the total area of the bars in the chart of both the functions φ and φ_1 is bounded by $100\,\%$. The increased weight of the first bars suffices to get a smaller estimate of 8.6 months for the development time. The standard deviation has a better value of 4.5. The probability of exceeding the estimate by at most the standard deviation is $28\,\%$. The results are the other way round for the worst case. The first bars lie below the corresponding bars of the initial example. Again there is a turning point. The worst case has a higher estimate of 13.5 months for the development time. The standard deviation is 7.6 and the error bound is $20\,\%$.

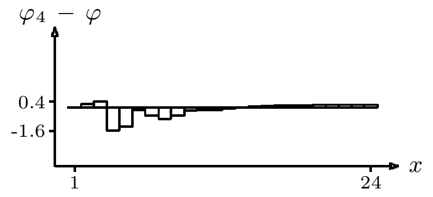
5.4 Different Base Probabilities

The model behaves similarly when changing the values of the base probabilities. If the base probabilities of the second team are replaced with those of the first team, the results improve. On the other hand, if the base probabilities of the third team are replaced with those of the second team, the results get worse. The other input data are fixed.









The chances of succeeding within two years are $96\,\%$ and $89\,\%$. The estimates for the development time are 9.9 and 11.8 months. The standard deviations are 5.7 and 6.8 months, the error bounds $25\,\%$ and $22\,\%$.

References

1. Abdel-Hamid : " Adapting, Correcting and Perfecting Software Estimates: A Maintenance Metaphor " , IEEE Computer 26-3 (1993) 20-29
2. Gray, Mac Donell : " A Comparison of Techniques for Developing Predictive Models of Software Metrics " , Information and Software Technology 39 (1997) 425-437
3. Kemerer : " An Empirical Validation of Software Cost Estimation Models " , Communications ACM 30-5 (1987) 416-429
4. Kemerer : " Reliability of Function Point Measurements: A Field Experiment " , Communications ACM 36-2 (1993) 85-97
5. Kitchenham, Taylor : " Software Project Development Cost Estimation " , Journal of Systems and Software 5 (1985) 267-278
6. Lederer, Prasad : " Nine Management Guidelines for Better Cost Estimating " , Communications ACM 35-2 (1992) 51-59
7. Low, Jeffery : " Function Points in the Estimation and Evaluation of the Software Process " , IEEE Transactions Software Engineering 16-1 (1990) 64-71
8. Matson, Barrett, Mellichamp : " Software Development Cost Estimation Using Function Points " , IEEE Transactions Software Engineering 20-4 (1994) 275-287
9. Padberg : " Ein wahrscheinlichkeitstheoretisches Modell für Softwareprozesse " , Technical Report (in German), Universität Saarbrücken 1997
10. Sallis, Tate, Mac Donell : *Software Engineering*, 1995
11. Shepperd : *Foundations of Software Measurement*, 1995
12. Shepperd, Schofield, Kitchenham : " Effort Estimation Using Analogy " , ICSE 18, International Conference on Software Engineering (1996) 170-178
13. Srinivasan, Fisher : " Machine Learning Approaches to Estimating Software Development Effort " , IEEE Transactions Software Engineering 21-2 (1995) 126-137
14. Wittig, Finnie : " Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort " , Australian Journal of Information Systems 1 (1994) 87-94

Thanks to the reviewers for valuable comments on the model.

New Address

Dr.-Ing. Frank Padberg
Fakultät für Informatik
Universität Karlsruhe
Am Fasanengarten 5
76131 Karlsruhe
Germany
padberg@ira.uka.de

This article was processed using the LaTeX macro package with LLNCS style