Facets of Software Development Represented by Model Systems: Analysis and Enhancement

Anca-Juliana Stoica

14th Int Forum on SCM/Focused Workshop, USC-CSE, Oct.1999

Contents

1. Introduction

Critical success factors for the software projects such as:

- setting realistic objectives for all stakeholders and successful project start
- tracking progress
- making smart decisions
- institutionalizing analyses in view of learning from previous experience

as well as the current and future trends in software systems like:

- ever-more complexity
- reuse-driven
- COTS-driven
- Internet-based

are arguments in favor of an integrated use of models (model systems) for software development.

Model systems are frameworks recently developed in order to integrate models associated to different facets of the software development. Examples are:

- Model Based (System) Architecting and Software Engineering (MBASE)[Boehm, Port, 1998]
- Model Based Software Engineering [Fisher et al., 1998]
- Software Engineering Institute's Model Based Software Engineering [Gargaro-Peterson, 1996]
- Honeywell's Model Based Software Development [Honeywell 1998]
- The Unified Software Development Process (USDP)[Jacobson, Booch, Rumbaugh, 1999].

Most of these models are oriented towards product and process models (sometimes also on their associated property models). An exception is the MBASE model system which integrates process, propery, product, and success models as well as interactions between them.

At a recent workshop on software research strategies [NSF Workshop, 1999] it was emphasized that an important focus in software science should be: i)explanatory/predictive models (process, product, user, interaction among models); ii)empirical validation technology and analytic methods.

Software engineering has a science base that needs development in areas such as:

- creation of integrative technologies and methodologies
- software evolution
- ability to predictably produce software in a dynamic setting
- (optimal) decision making support regarding changes in an efficient and reliable manner.

The focus in this paper is:

- to show how facets of the software development are related in different approaches such as the MBASE model system and the USDP generic process framework
- to present an enhancement of the MBASE success models with a reasoning and decision framework (DF) in which optimal decisions in software development are based on formal dynamic models related to cost, schedule, process, and risk.

DF is applicable especially in the early stages of the software development process when major decisions have to be taken and major risks have to be managed.

2. Models and Model Systems

Model definition

- Traditionally a model has been defined as an abstract description of a system to be developed
- More recently a viewpoint has been added to this definition, so the model is a semantically closed abstraction of a system to be developed from a certain viewpoint and a certain level of abstraction.

The views

- cope with the complexity of software systems and represent abstractions of relevant information for the model
- address one or more concerns of a system stakeholder defined as an individual, a group, or an organization that shares concerns or interests in the system (e.g. developer, user, customer, manager, financing agencies, etc).

Frameworks are needed to integrate and analyse views (models). Redundant information is used to verify consistency and completeness between views.

Software system

- Traditionally has been associated with all artifacts used to represent it in human or machine readable form such as: models, source code, executables, documentation. Artifacts are considered to be any kind of information created, produced, changed, or used by the system developers.
- From a recent model system perspective, a software system is associated with collection of models representing different views (which are defined in both the above ways) of the system.

Because of the broad definition of stakeholders the model range has been expanded in some model systems to include other models in addition to the product and process models such as success and property models. The model definition becomes more general, namely: a pattern of something to be made, a decription or an analogy used to visualize and reason about the system to be developed and its likely effects. The sources of these models are also very diversified, as it is shown in [Boehm, Port,1998] for example.

A formal modeling language like UML [Booch, Rumbaugh, Jacobson, 1998] is used for software design to ensure that each model is consistent with itself and other models. The UML models are: use cases (elicit requirements), class diagrams, inheritance, interaction diagrams (collaboration and sequence), package diagrams, state diagrams, deployment diagrams, patterns, and relationships (association, aggregation, dependency, multiplicity, and navigation).

Other formal languages like ADLs are used for coarse-grained software architecture: components, connectors, configurations.

We define a **model system** as a collection of models and relationships between them. A model system will therefore contain all relationships and constraints between model elements contained in different models. A graphical representation of a model system is given in Fig.1.

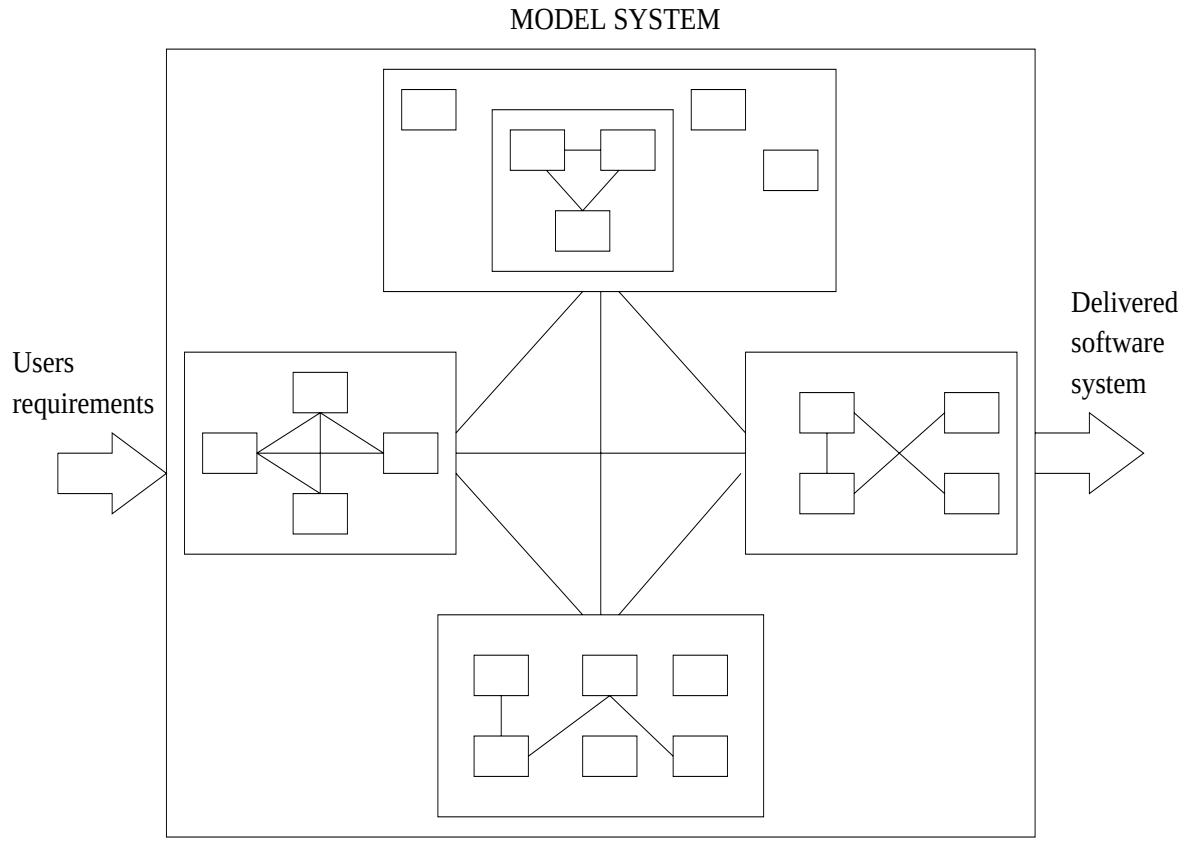

MODEL SYSTEM
Users
requirements
Delivered
software
system

Figure 1:

Model systems

- define a notational and a semantical integration
  - what information
  - how it can be exchanged
  - how to detect inconsistencies
- are closely related to decision making:
  - selecting them is one of the most important decisions the development team makes
  - the decision making process is used for evaluation and analysis of the other model implementations
- have benefits
  - model the software before building it
  - allow reasoning, optimization, and decision analysis.

Examples of Model Systems

Example 1: **The USDP model system**

- oriented towards product and process models
- consists of: use case, analysis, design, deployment, implementation, and test models which are related to represent the system as a whole
- there are trace dependencies, backward and forward links to other models (Fig. 2)
- traces connect elements in one model to elements in another model but add no semantic information
- the ability to trace improves the understandability and change propagation in software development.

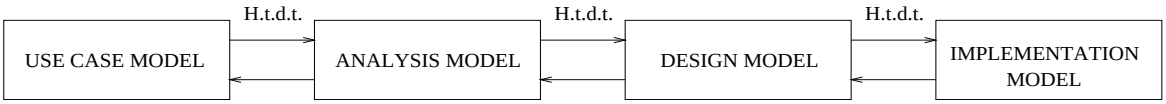


Figure 2:

In Fig.2 each of the model can be further represented using other models or model elements. Examples are:

- use-case model (UCM) for representing the functional requirements (Fig.3)
- design model (DM) represented in Fig.4 (there is an hierarchy of elements in this model).

Use-case (functional requirements) model

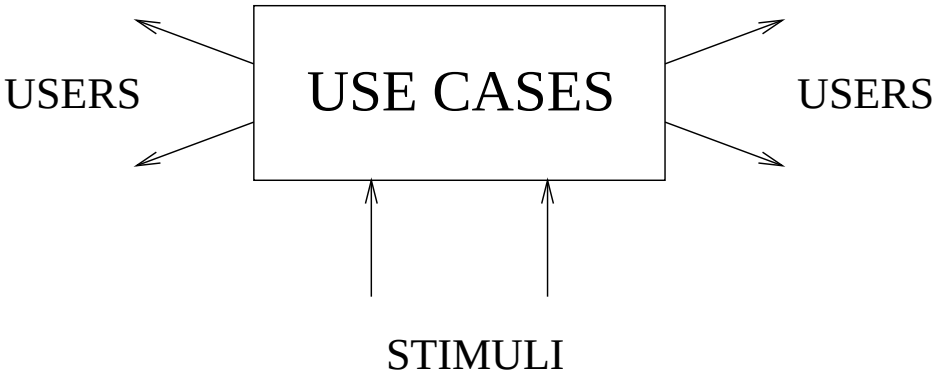


Figure 3:

Design model

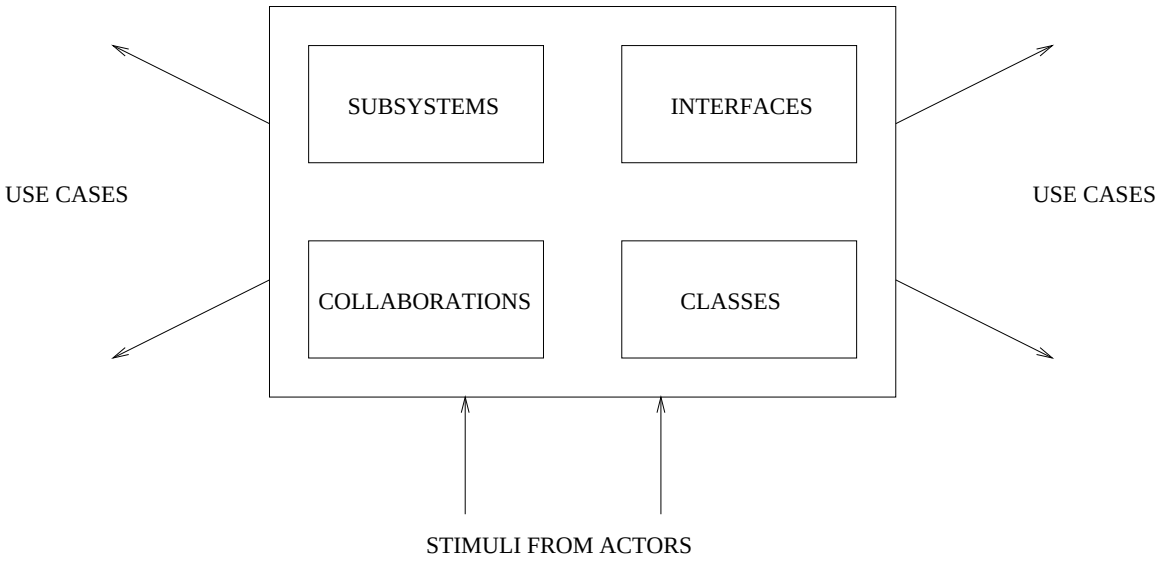


Figure 4:

DIAGRAMS −> WORK-FLOWS	Use case	Class	Compo-nent	Deploy-ment	Seq.	Collabo-ration	State chart	Activity
Use Case Model	X				X		X	X
Analysis Model		X			X	X	X	X
Design Model		X			X	X	X	X
Deployment Model				X	X	X		
Implemen-tation Model			X		X	X		
Test Model	X	X	X	X	X	X	X	X

Table 1:

Example 2: **The MBASE model system**

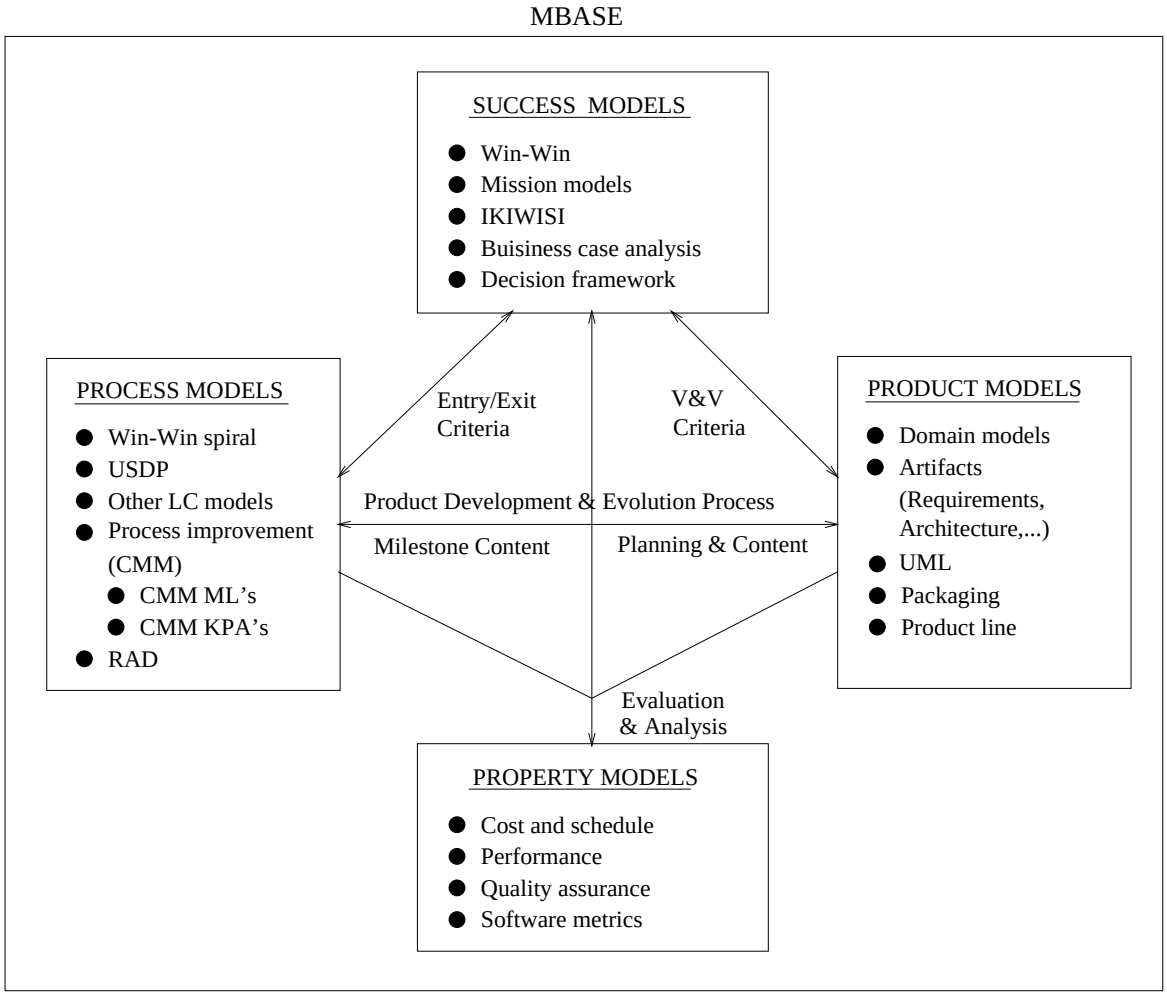


Figure 5:

- building a software system is a process of model building using different models to describe all the different perspectives of the system
- models need:
  - integration
  - consistency
  - mutual enforcement
  - compatibility,

  implemented by using analyses, decision tables, checklists) [Boehm, Port, 1998]. Integration can also be done by view integration frameworks which include logical operations such as mapping, transformation, differentiation [Egyed, 1999].

3. Generic Software Process

A. Semantics

- model systems are associated to a generic software process
- a **generic software process** is complete set of generic activities applicable to all software projects (regardless of their size and complexity), and their associated results (artifacts)
- a generic process has to be instantiated in a particular set of circumstances (a particular process)
- a **process instance** is equivalent to a **project**. Breaking down the generic process into finer-grain activities is a project specific task.

In an **incremental development** the generic software process has

- a time dimension - how the time is divided
- a product growth dimension - how growth in the product is achieved.

Each product consists of a series of releases to customers (external releases).
An external product release consists of:

- i) source code (components)
- ii) manuals and associated deliverables
- iii) product, process, and domain models.

A **product release** to the customers is achieved in four major steps:

- **Inception** - defines the vision of the end product and the business case
- **Elaboration** - detailed specification of most use cases and architecture
- **Construction** - product is built
- **Transition** - product moves to beta release.

A major step is terminated with a **milestone**. A milestone indicates:

- the availability of a set of artifacts (models or documents in a prescribed state)
- that crucial decisions are made
- the progress monitoring.

Main milestones are: **LCO** (Life Cycle Objectives), **LCA** (Life Cycle Architecture), **IOC** (Initial Operational Capability) at the end of: Inception, Elaboration, and Construction major steps.

Major steps are further divided into **substeps**. A substep is like a **mini-project** where a particular **process model** can be applied, such as: **waterfall** or **spiral**.

A substep adds an **increment** (growth in the product), defined as the difference between two succesive internal releases. A mini-project deals with a group of use cases and the most important risks. It ends with a baseline.

A **baseline** is a set of models that represent a system which has reached a particular state. **Configuration management** deals with maintaining consistent and compatible versions of all artifacts in a baseline.

The end of a substep is a closure with the development team indicating that: i)all software is integrated; ii)can be internally released.

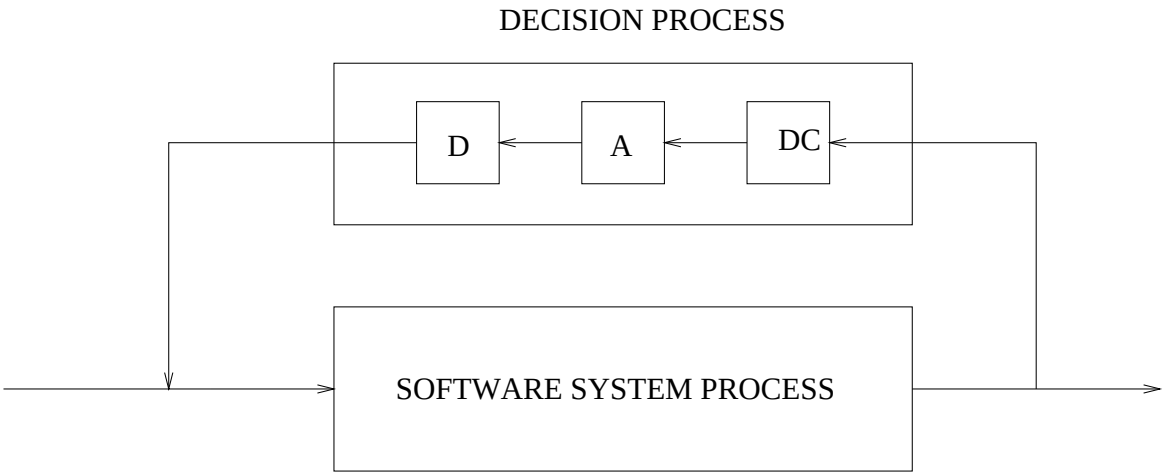
The above semantics applies to generic process models such as **MBASE** and **USDP**. There are also differences between them. Basically

- **MBASE** models cover a wider spectrum: process, product, property, success models. The time subdivision is the spiral cycle with the spiral process model and the spiral cycle template. Milestones are defined in terms of MBASE elements or views: OCD, RDP, SSRD, SSAD, LCP, FRD [Boehm, Port, 1999].
- **USDP** models cover process, product, domain models. The time subdivision is the iteration with the waterfall process model and the workflows carried out for the: requirements, analysis, design, implementation, and test. Workflows structure: workers, artifacts, interactions (for creation, production, and use of each others' artifacts) [Jacobsson, Booch, Rumbaugh, 1999].

B. Decision Process

The decision under risk process accompanies the software process and consequently it should be integrated into a model system. Using a formal model for representation it allows for dynamic optimization which can lead to a better foundation for the software development especially in the early stages when costs, risks, information available or acquirable, performance, scheduled time, have to be integrated.

As shown in the previous section, there are models to represent the software process but there are no models to represent the associated decision under risk process in view of reasoning and optimization. A suggested framework for this problem is the DF approach presented in section 5. Here a graphical representation of the software system and decision processes is given in Fig.6.



D - Decision
A - Analysis
DC - Data Collection

Figure 6:

In both the system and the decision processes an important problem is the data collection problem. In general data collection (DC) refers to: software metrics, staffing, inter- and intra-group communications, quality, or process data. The data has to be analysed (A) in order to feed the decision (D). Decisions refer to major problems such as: selecting the models for a system by the development team, selecting new technologies, use cases, architecture.

A formal(mathematical) representation of the DF is given in section 5.

Considering the critical success factors from section 1, it follows that the most appropriate strategy for the present and predictably future software processes is the evolutionary development with the following features: iterative, COTS-driven, reuse-driven, and risk-driven.

The risk-driven software development and evolution process is substituting the former document- and/or code-driven, so iterations in the software process are organized as to achieve risk reduction and to baseline architectural functionality.

The primary risk objectives for the software and decision processes are to nail the risks in early iterations. This is where the DF approach applies as well as to the analyses in view of risk reduction in the inception and elaboration phase iterations. Later, in the construction phase the risks for the most part have been reduced to a routine level, meaning that they yield to ordinary development practices.

Tables 2 and 3 show the main phases of the product cycle: inception, elaboration, construction, and transition, their associated milestones: LCO, LCA, IOC, and some important facets such as:

- essential objective: what?
- essential operational activities: how?
- level of estimation for: cost, effort, schedule, and quality
- essential analyses to be performed
- risk strategy.

Phase of the LC & milestone	Essential obj.: What?	Essential oper. activ.: How?	Level of estim. for: cost, effort, schedule, and quality	Essential analyses	Risk strategy
INCEPT. (LCO)	Viability	Moving from a key subset of reqs to a candid. architect.	Initial, broad limits	Business case init. (project has to be econ. worth within broad limits)	Identify and resolve critical risks
ELAB. (LCA)	Build the system in an econ. framework	Spec. of most use cases Extend candid. architect. to exec. baseline Project plan for CONSTR.	Narrow limits to justify a business bid	IRR Finalize the business case	Grad. reduce remaining risks by contr. iterations

Table 2: (to be cont.)

Phase of the LC & milestone	Essential obj.: What?	Essential oper. activ.: How?	Level of estim. for: cost, effort, schedule, and quality	Essential analyses	Risk strategy
CONSTR. (IOC)	System capable of init. oper. in the users' environm.	Series of iter. (builds & increments) so that syst. viab. is evident in exec.form	Revise for even increased accuracy	Performance Usability Availability Cost effectiveness	Red. to a routine level (they yield to ordin. dev. practices)
TRANS.	A system that achieves final oper. capab.	Modify prod. to allev. probls. not ident. in prev. cases Correct defects	TBD	TBD	TBD

Table 3: (end)

There are several categories of **technical risks** and associated decisions that can be instantiations of the decisional framework:

- **architectures**, like robust architectures where reusable components fit in make or buy decisions
- **new technologies**
- **building the right system** which supports mission and users. This translates to: i)find the most important functions and implement them early; ii)prioritize use cases according to their importance for meeting customers' needs and performance requirements; iii)prioritize the use case implementation order based on their risk potential; iv)establish the correlation between requirements and their risks in the early phases; v)determine the selected use cases impact on the architecture.
- **performance risks**, like: response time of a use case, number of concurrent use case instances/hour, etc.

Risk identification, analysis, and prioritization are the first steps of risk management, which correspond to risk assesment [Boehm, 1989]. After risk assessment, the next steps are: risk management planning, risk resolution, and risk monitoring, which are all dealing with risk control.

It follws that there are several choices in dealing with risks such as: i)avoid by replannig or changing requirements; ii)confine or restrict; iii)mitigate or try it out which implies materialize a contingency plan and retire; iv)monitor using a monitoring mechanism, set it up and execute it.

4. Analysis and Extension of a Model System

In this section we analyse the **MBASE** model system and suggest an extension of it with a formal decisional framework (**DF**).

MBASE conceptual modeling is a special case of general knowledge representation for expert systems. It is used for the development of software-intensive systems by using models to visualize and reason about prospective systems and their likely effects.

The **models** satisfy the general definition of a model presented in section 2 and consequently cover a wide spectrum: laws or regulations (government acquisition processes), business models, hard-won or recent experience, operational effectiveness, generic process models, domain models, software artifacts, software paradigms, cost/schedule/performance models, software metrics. They are divided into: **success**, **product**, **process**, and **property** models and are grafically represented in Fig.5, section 2.

When they conflict with other, a **model clash** occurs. Examples of model clashes and their remedies are given in [Boehm, Port, 1999].

The principal **system problems** associated with these models are: integration, consistency, mutual enforcement, and compatibility (by analyses, decision tables, checklists).

Model integration can be done using a variety of approaches such as:

- domain-driven model integration, applicable to families of projects in relatively mature or bounded domains, like multimedia archive domain
- software process model decision tables
- model compatibility analyses such as QUARCC [Boehm, Inn, 1996], SAAM [Kazman et al., 1994], Expert COCOMO [Madachy, 1995] which are good for finding compatible models
- model-clashes tables as checklists
- knowledge-based toolset to provide advice on avoiding model clashes
- view integration framework using logical operations such as: mapping, transformation, and differentiation [Egyed, 1999]

We propose an extension of **MBASE** model system with a decisional framework (**DF**)as part of the success models subsystem.

DF is a formal(mathematical) dynamic decisional framework that allows reasoning about the optimal sequence of decisions over a time horizon based on examining possible alternatives, their costs, associated risks, and estimated benefits. Using the framework it is also possible to apply inductive reasoning techniques based on Bayesian methods to analyse the effects of waiting and designing experiments to get more information and consequently learn from them. The semantics of MBASE model system (including DF) relations is presented in Fig.7.

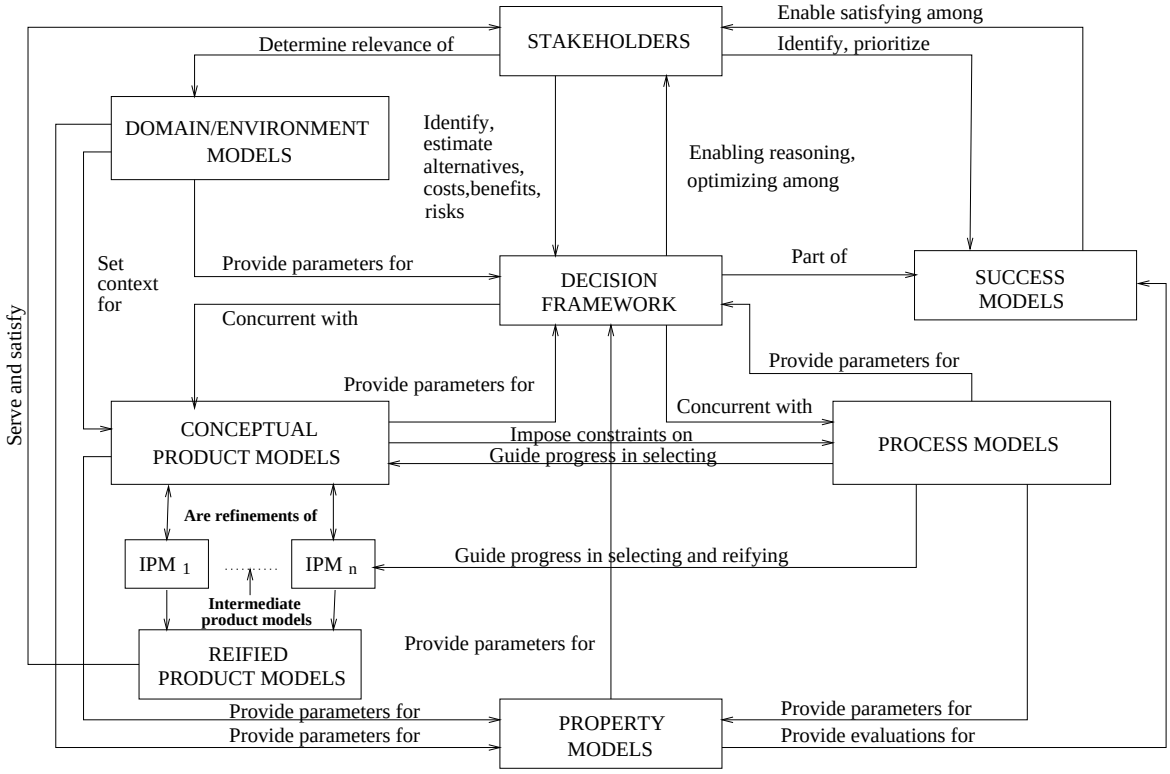

STAKEHOLDERS
Enable satisfying among
Identify, prioritize
Determine relevance of
DOMAIN/ENVIRONMENT MODELS
Identify, estimate alternatives, costs,benefits, risks
Enabling reasoning, optimizing among
Set context for
Provide parameters for
DECISION FRAMEWORK
Part of
SUCCESS MODELS
Concurrent with
Provide parameters for
Provide parameters for
Concurrent with
Serve and satisfy
CONCEPTUAL PRODUCT MODELS
Impose constraints on
Guide progress in selecting
PROCESS MODELS
Are refinements of
IPM 1
IPM n
Guide progress in selecting and reifying
Intermediate product models
Provide parameters for
REIFIED PRODUCT MODELS
Provide parameters for
Provide parameters for
PROPERTY MODELS
Provide parameters for
Provide evaluations for

Figure 7:

An example of a MBASE model system instantiation which includes DF is given in Table 3.

Primary success model	Demo scalable multicast communication system on the Internet in "X" months
Secondary success model	DF with time horizon = "X"
Key stakeholders	Entrepeneurs, venture capitalists, customers
Key property models	• Schedule estimation • Performance evaluation parameters: maintainability, usability
Process model	Design-to-schedule
Product model	Domain constrained by schedule; architected for essential features to meet schedule.

Table 4:

Remarks:

- DF is an enhancement of the MBASE success models because it relates costs, schedule, and process/product decisions in a dynamic optimal mode
- a cost and schedule property model can be used for the evaluation and analysis of the consistency between system's product, process, and success models
- a product line is initially focused on product models such as: domain models, product line architectures, with process and property models subsequently explored to perform business-case and dynamic decision analyses of the most appropriate breadth of the product line and the timing for introduction of new products
- usually property models are used to verify the consistency of the other models at the process milestones LCO and LCA.
- DF can be used together with SEDA [Boehm, Port, 1998] to provide optimal decision suggestions to software decision-makers in order to revise the plans and perform software evolution (see Fig. 8)

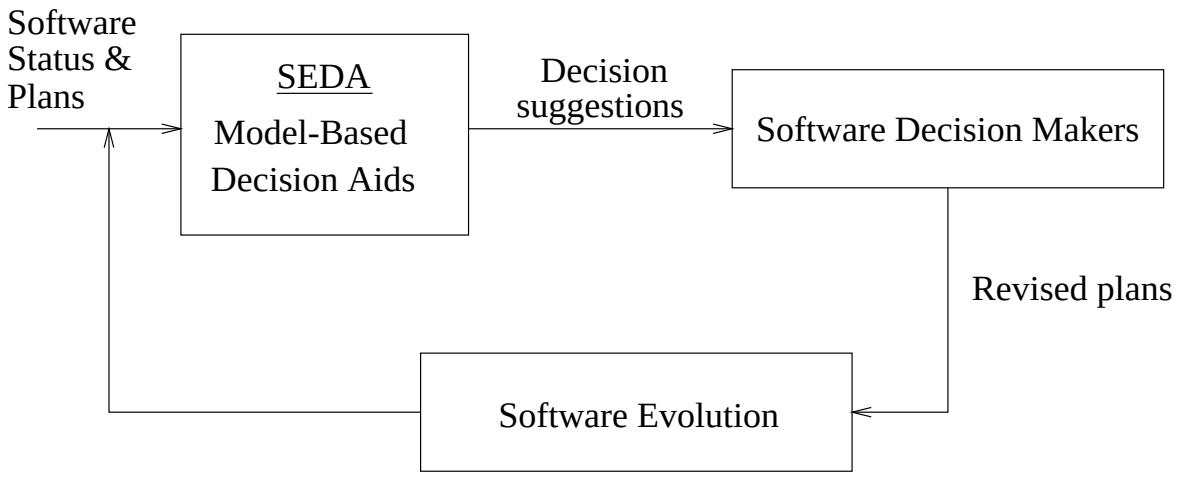

Software
Status &
Plans
SEDA
Model-Based
Decision Aids
Decision
suggestions
Software Decision Makers
Revised plans
Software Evolution

Figure 8:

5. Decisional Framework (DF)

- focuses on common characteristics of the software process decision making amenable to be used in a theoretical decisional framework based on dynamic predictive models of the software process as value-added in face of an uncertain world.
- meta-model used to capture the common elements of the software decision problem (SDP) complexity but gives the latitude to use techniques that are appropriate for a particular case
- modeling language notation represented by multi-sequential decision trees (for inferring an optimal sequence of decisions over a time horizon)
- analyzes deferring decisions
- uses information matrices and Bayesian analysis as a mode of inductive reasoning
- dynamic economic criterion which accounts for the time value of money
- backward dynamic programming to get the problem solution.

The **decisional framework** is used for:

- i) identifying and quantifying influential factors on various issues of interest, for the purpose of understanding and controlling the process;
- ii) reasoning about concepts and alternatives;
- iii) analytical evaluation of the optimum strategy under economic constraints.

A. Decision under Risk Models

Decision theory deals with modeling the general structure of a decision process, characterized by the existence of the following :

- a single decision-maker or a group
- a problem formulation
- a set M of possible alternatives (or options)
- a set N of the states of the world or possible context-dependent conditions which can occur in a decisional problem
- a set P of decision criteria or decision maker's objectives
- the time set T (if the model is dynamic)
- the set X of anticipated consequences in case of each combination of: alternatives (i), states of the world (j), criteria (k), and time (t): $X = \{x_{ijk}\}t$, where $i \in M, j \in N, k \in P, t \in T$.

The decision methods are applicable if the folowing hold:

- the problem is formalized
- the relationships and dependencies are expressed by symbols to facilitate the use of mathematical methods
- the actions' results are measurable.

B. Decision Methods in the Software Process

The characteristics of the software process formalized as a decision frameworks are:

- Dynamic and predictive, based on common features of specific processes
- The possibility of determining the time horizon (e.g. by "design-to-time" or by analytic models for schedule computation)
- A priori or a posteriori determination of risks (probabilities of unsatisfactory outcomes). The a priori determination is based on subjective estimation (individual or group consensus techniques), whereas the a posteriori evaluation is based on experiments such as: prototypes, simulations, surveys,...
- Software process is a value-added process, where investments in software assets like: COTS, reusable components, architectural options, operational options (scenarios, task models, major data structures, control structures, algorithms), program generators, information hiding interfaces are contingent on future outcomes and have to be optimized dynamically.
- The software process as a dynamic, optimized for value-added decisional process is closely connected to

information theory, since experimentation creates the information structure used to refine the subjective probability estimates of expert analysts to conditional probabilities

- Use of Bayesian analysis as a mode of inductive reasoning: i)expert judgement information used in a logically consistent manner in making inferences; ii) Bayes' theorem allows the computation of a posteriori probabilities of events; iii) the transformation of a priori (initial) probability values into a posteriori values can be assimilated to a learning process, because we design an experiment and learn from it (i.e. we get more information based on it)
- The decision rule based on a dynamic economic criterion (V) accounts for the expected values of benefits and for the costs discounted for the time value of money and contingent on what kind of future states of the world actually unfold.

We are looking for the optimal timing and optimal investment alternative(s) in software assets based on the above decision rule. The decisional techniques (methods) used to solve this type of problems, namely software process decision problems will be based on a mix of:

- decisional utilities for expected value calculations
- dynamic programming and indexing techniques for the V criterion optimization given a time horizon
- analytic models for the schedule estimation and specification of the time horizon used in the dynamic optimization
- decision methods under risk (the probabilities of states of the world estimated by group consensus techniques, averaging techniques, or expert evaluation): dynamic decision trees, Bayesian decision technique and the associated value of information.

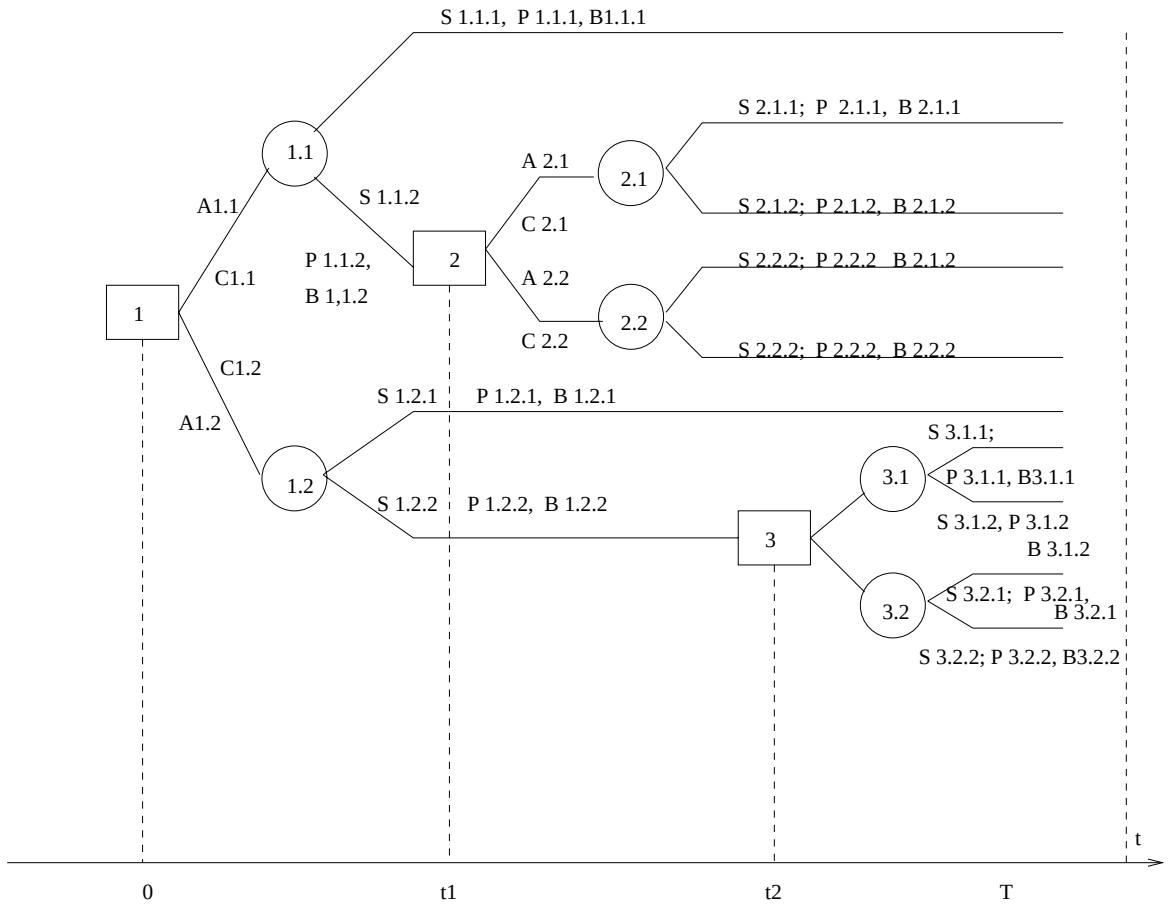
C. Mathematical Background of the Software Decision under Risk Process

i) Decision Trees

The software decision process (SDP) is modeled by means of a discrete decision tree of finite depth T, where T is the schedule, or the maximum number of future time steps to be considered in the framework.T is the time horizon. The decision treees are used to represent a sequence of dynamic decisions (e.g. decisions made at different time instances) or a decisional process. A decision tree is a directed graph open to the future (see Fig.9).

The root at depth 0 represents the present time. There are two types of nodes:

1. Decision nodes - the choice is made by the decision maker, according to a predermined economic criterion (V).
2. Event nodes, to which we associate the states of the world (events which occur with a certain probability (known or estimated, a priori or a posteriori), and upon which the decision maker has no control).



Time : t	0	t1	t2
Step : k	1	2	3
Depth : d(k)	0	t1	t2

Legend
Decision Node
Event Node

Figure 9:

The decision tree (DT) can be:

- monosequential, if there is only one decision instance
- multisequential, if there is a sequence of decisions made at $t = 0$, $t = t_1$, $t = t_2$ in Fig 1).

For each alternative j represented by a branch from each decisional node i at stage k we use the following notations:

- $E(B_{ij}(k))$ - the expected benefit when we choose alternative j from node i at stage k discounted for the time value of money
- $C_{ij}(k)$ - the alternative j (from node i) estimated cost at stage k
- $d(k)$ - the depth of a node evaluated at stage k.

Nodes at depth $d(k)$ represent possible decisions to be made at stage k. For example, in Fig.9 we have 3 stages: 1, 2, 3 and the corresponding times are: 0, t1, t2. The stages are the time instances when we have to decide for an optimal alternative.
We use the following formulas for discounting future cash flows:

1. Single cash flow

$$PV(F, r, n) = FD^n \tag{1}$$

2. Multiple cash flows

$$PV(F, r, n) = F(\frac{1 - D^n}{1 - D}) \tag{2}$$

where:
PV - present value
F - future cash flow
n - the number of future time instances
$D = 1/1 + r$, the discount rate
r - the interest rate

ii) The Optimization Problem

- Traditionally the optimization problem was applied at the computer resource level like: CPU, memory, files, et al.
- Currently it becomes more and more important to apply it at a higher level, like architectural and design levels, or the levels at which the DF applies.

There are several reasons for this:

- the demand for a quantitative assessment of software artifacts and techniques, which is needed to assess risk and find optimal trade-offs among software quality, cost, and schedule especially during the inception, elaboration, and construction phases
- when using COTS there are 2 effects to consider: i) a level of indirection in the execution of a program and data conversions between components which generates overhead; ii) three-tier software development adds network latency and its own data communication overhead.

It follows that the burden of optimizing has shifted away from programming to the modeling, design, and architecting software products and their associated decisions.

In the DF decisions are made considering the time horizon $[0, T]$ using dynamic programming in which computations start at the last stage and then proceed backward to stage 1. At the last stage n we choose at each node i the alternative j that maximizes V:

$$\begin{aligned} V_i(n) &= \max_{j \in J_i} \{V_{ij}(n)\} \\ &= \max_{j \in J_i} \{E(B_{ij}(n) - C_{ij}(n)\}, i \in I_n \end{aligned} \tag{3}$$

where the expected benefit $E(B_{ij})(n)$ is the probability-weighted sum of possible outcomes computed as follows:

$$E(B_{ij}(n)) = \sum_{l \in L_{ij}} P_{ijl}(n) B_{ijl}(T) \left(D^{d(T)-d(n)} \right), i \in I_n, j \in J_i \tag{4}$$

L_{ij} is the set of outcomes associated to the event node successor of decisional node i on the path containing alternative j

$P_{ijl}(n)$ is the probability of outcome l satisfying the same above conditions

Stepping backward in time in the DT we compute $V_i(k)$

at depth $d(k)$, stage k ,and node i as follows:

$$\begin{aligned} V_i(k) &= \max_{j \in J_i} \{V_{ij}(k)\} \\ &= \max_{j \in J_i} \sum_{l \in L_{ij}} P_{ijl}(k)(B_{ijl}(k) \\ &+ V_{p_{ij}}(k+1)(D^{d(k+1)-d(k)})) \\ &- C_{ij}(k), i \in I_k, k = 1, \ldots, (n-1) \end{aligned} \tag{5}$$

($B_{ijl}(k)$ is the discounted benefit relative to time at stage k, and $V_{p_{ij}}(k+1)$ is discounted for a single cash flow to be obtained at stage $k+1$ relative to stage k. The procedure determines the optimal decision sequence and optimal investment policy for a decision time horizon $[0, T]$. The probabilities $P_{ijl}(k)$ and the benefits $B_{ijl}(k)$ are expert judgement based on main sources of cost savings (benefits) and on analysis factoring in technical capability trends, technology transition effects, and normal commercial trends. Individual opinions can be aggregated in a group estimation using e.g.: i)averaging techniques; ii)group decision techniques; iii)the Delphi method. It assists the software LC planning to sequence the work so that the most important decisions on: new technologies, architecture, use cases (identification, prioritization based on importance in order to deal with them, correlation requirements and risks, impact on architecture), can be made early.

iii) Probabilities and Information

Information is used to improve the decision quality, so its value is determined with respect to the operational decisions in the software process using Bayesian statistical analysis. Probability measures our level of knowledge about a fact or an event, and not its real state, which is particularly important in solving decision problems. Consequently, to evaluate the probabilities we use the available information at a certain time. The problem is to express the available information in an usable form, one possibility being to code information as a probability.

Let X_i be a complete event system. We assume that we know the initial a priori probabilities $P(X_i)_{i\in M}$. We perform an experiment which generates the results Y_j. Assuming that the conditional probabilities $P(Y_j/X_i)$, $i \in M$, $j \in N$ can be determined, Bayes's formula evaluates the probabilities of the complete event system $P(X_i)_{i\in M}$ by knowing the results of the performed experiment. The new probabilities will be $P(X_i/Y_j)$, $i \in M$, $j \in N$ and are called final or a posteriori:

$$P(X_i/Y_j) = \frac{P(X_i)P(Y_j/X_i)}{\sum_{i\in M} P(X_i)P(Y_j/X_i)}, j \in N \qquad (6)$$

The modified values of the a posteriori probabilities (as a function of the experiment results) reflects the experiment importance or its informational value.

iv) Further Refinement of the Decisional Problem

We assume the decisional problem formalization where:
X_i - state i of the world, $i \in M$.
A_j - alternative j generated by the experiment's results
$P(X_i)$ - a priori probability of state of the world X_i
The payoff matrix is $[V_{ji}]$, $i \in M$, $j \in N$, where V_{ji} is the payoff when choosing A_j, if the state of the world is X_i. Let $U_{ji} = U(A_j, X_i)$ be the utility (in the sense of Von Neumann-Morgenstern) of the pair (A_j, X_i). Performing the above mentioned experiment generates an information structure, called the information matrix $[q_{ij}] = [P(Y_j/X_i)]$, where $Y_j, j \in N$, is the message j transmitted to the decision-maker by the information $I = 1, \ldots, n$ and q_{ij} is the conditional probability of message j, given that the world is in state i. The information reduce uncertainty. If the message codes are a priori known the uncertainty is reduced to zero after the message was received. Partial information also reduces uncertainty. The use of information to reduce uncertainty can provide a conceptual understanding of the information value.

We denote by $A = \alpha(Y)$ a decision rule which associates an action to each of the possible messages. The expected value of A_j if message Y_j was received is:

$$E(V(A_j/Y_j) = \sum_{i \in M} P(X_i/Y_j)U_{ji} \tag{7}$$

In (7), $P(X_i/Y_j)$ is computed with the Bayes' formula (6).The expected value of the imperfect information communicated by the experiment is:

$$E(V) = \sum_{j \in N} P(Y_j)E(V(\alpha(Y_j)/Y_j)) \tag{8}$$

$$= \sum_{j \in N} P(Y_j) \sum_{i \in M} P(X_i/Y_j)U(\alpha(Y_j), X_i) \tag{9}$$

In (9) because the a priori probabilities and the information matrix elements $P(X_i/Y_j)$ are known, we can calculate the probabilities $P(Y_i)$ using the total probability formula:

$$P(Y_j) = \sum_{i \in M} P(X_i)P(Y_j/X_i), \tag{10}$$

and the a posteriori probabilities with (6).
Bayesian analysis can also be used in a decision tree to evaluate nodes and decision making in cases for example when there is a possibility of waiting, performing an experiment to get more information, and choosing an alternative based on applying the $\{maxE(V)\}$ criterion face of future possible outcomes.

An Example (analyzing the effects of waiting before deciding)

Suppose that at time $t = 0$, it is possible to finance a novel design solution by stakeholder Y with C_1, or after $t = t_{wait}$ to finance it by stakeholder X partially with C_2 or totally with C_3, where $C_1 < C_2 < C_3$.
In order to determine the technical compatibility between the proposed solution and its objectives, X can perform an experiment and its results are communicated via the information matrix.
After the waiting time it is still possible to consider the Y's offer.

Considering the probabilities of all future possible outcomes the problem is to find the optimal sequence of decisions to apply in this case.

Solution: We compute the expected value of the event node corresponding to the decision to experiment, which is made after waiting for $t = t_{wait}$:

$$E(V_{exp}) = \sum_{k \in R} P(Y_k) \max_{j \in J_{D_k}} \left(\sum_{i \in S_j} P(X_i/Y_j) C_{i_j} \right) \quad (11)$$

where

- R is the experiment result set
- D is the decision set (d_1-wait, d_2-Y finances now, d_3-X finances after wait, d_4-Y finances after wait)
- S is the state of the world set

The expected value of the decision *not* to experiment after waiting is:

$$E(V_{no-exp}) = \max_{j \in J_D} \left(\sum_{i \in S_j} P(X_{i_j}) C_{i_j} \right) \qquad (12)$$

where

- $P(X_{i_j})$ is the probability of state X_i, $i \in S$ in case of applying decision d_j after wait
- J_D is a subset of the decision set after wait
- S_j is a subset of the state of the world set corresponding to decision j after wait
- C_{i_j} is the consequence of decision j and state i

We choose between *experiment* and *do no experiment* at $t = t_{wait}$:

$$E(V_{wait}) = \max\{E(V_{exp}), E(V_{no-exp})\} \qquad (13)$$

and then between *wait* and *no wait* :

$$E(V_0) = \max\{E(V_{wait}), E(V_{no-wait})\} \qquad (14)$$

at $t = 0$.

D. Computational Examples *i) Example 1: Software reengineering)* A design team has to take decisions for the next 10 months about restructuring a software system in order to clean up its modular strucure which has degraded as result of maintenance over time. Restructuring offers the possibility to make the system easier to understand and to maintain.
The alternatives for restructuring are:

- A1.1 External CASE tools for remodularizing
- A1.2 Internally developed CASE. If this is not successful, then after 2 months
  - A2.1 Licence from a vendor
  - A2.2 Use better design methods.
- A1.3 Restrucure on a smaller scale (a subset of programs) and if there is a high demand, then after 5 months:
  - A3.1 Switch to wholescale
  - A3.2 Continue small scale restructuring, with no additional cost.

Solution

We assume that costs and benefits given for a time interval like 10, 8, 5 months from Fig. 10 representing the decision tree are discounted.

We apply dynamic programming to find the optimum decisional sequence and the associated investment costs, considering a 10 months time horizon.

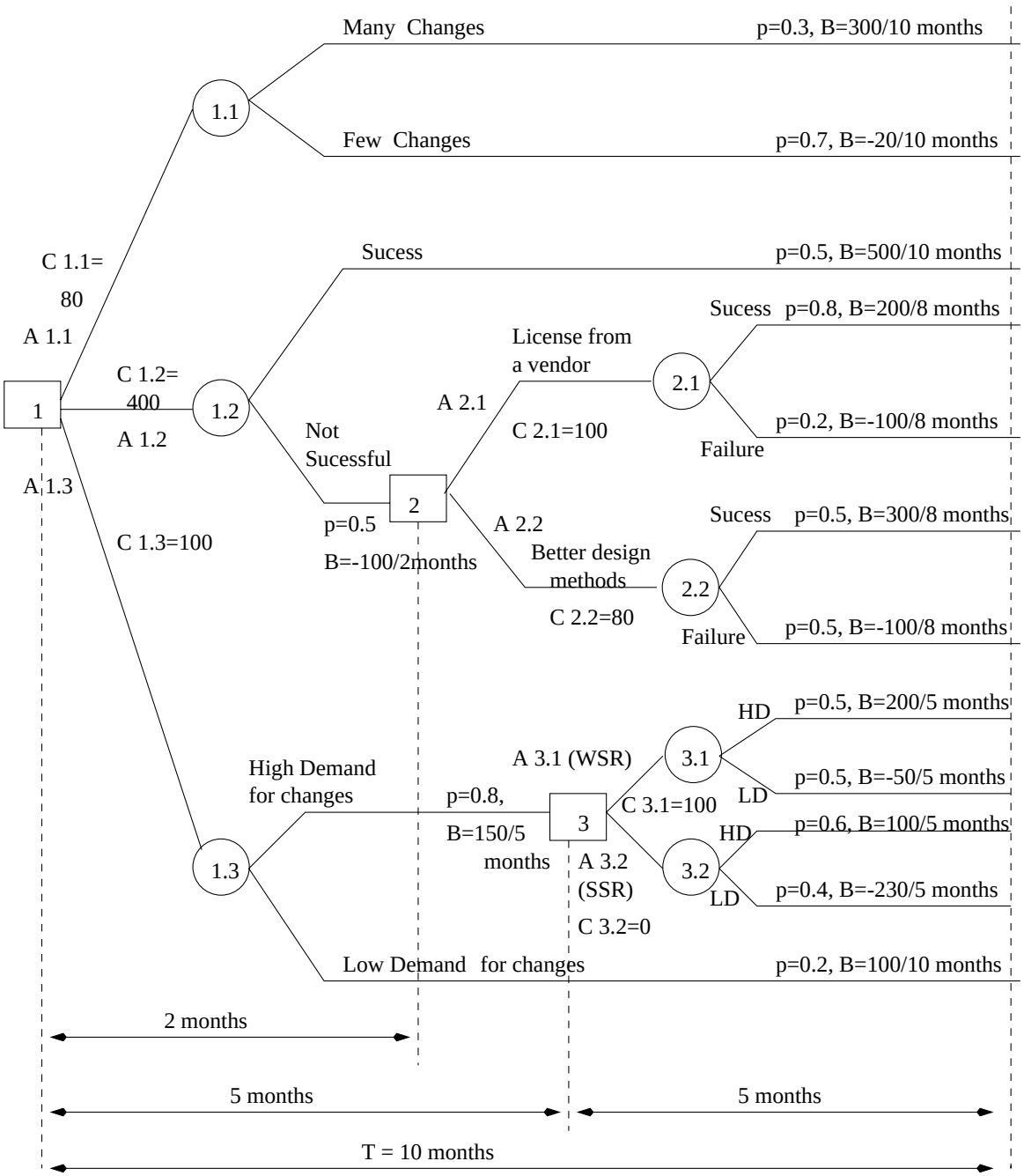

Many Changes
p=0.3, B=300/10 months
1.1
Few Changes
p=0.7, B=-20/10 months
C 1.1=
80
A 1.1
Sucess
p=0.5, B=500/10 months
Sucess p=0.8, B=200/8 months
License from
a vendor
2.1
C 1.2=
400
A 1.2
1
1.2
A 2.1
C 2.1=100
p=0.2, B=-100/8 months
Failure
Not
Sucessful
A 1.3
2
p=0.5
B=-100/2months
A 2.2
Sucess p=0.5, B=300/8 months
C 1.3=100
Better design
methods
2.2
C 2.2=80
Failure
p=0.5, B=-100/8 months
HD
p=0.5, B=200/5 months
A 3.1 (WSR)
3.1
High Demand
for changes
p=0.8,
B=150/5
months
3
C 3.1=100
LD
p=0.5, B=-50/5 months
HD
p=0.6, B=100/5 months
1.3
A 3.2
(SSR)
C 3.2=0
3.2
LD
p=0.4, B=-230/5 months
Low Demand for changes
p=0.2, B=100/10 months
2 months
5 months
5 months
T = 10 months

Figure 10:

Stage 3

$$V_3(3) = \max\{V_{3.1}(3), V_{3.2}(3)\}$$

$$V_{3.1}(3) = E(B_{3.1}(3)) - C_{3.1}(3)$$

$$E(B_{3.1}(3)) = (0.5)(200) + (0.5)(-50) = 75/5 months$$

$$V_{3.1}(3) = 75 - 100 = -25/5 months$$

$$V_{3.2}(3) = E(B_{3.2}(3)) - C_{3.2}(3)$$

$$E(B_{3.2}(3)) = (0.6)(100) + (0.4)(-230) = -32/5 months$$

$$V_{3.2}(3) = -32/5 months$$

$$V_3(3) = \max\{-25, -32\} = -25/5 months$$

We choose $A_{3.1}$ (wholescale restructuring).

Stage 2

$$V_2(2) = \max\{V_{2.1}(2), V_{2.2}(2)\}$$

$$V_{2.1}(2) = E(B_{2.1}(2)) - C_{2.1}(2)$$

$$E(B_{2.1}(2)) = (0.8)(200) + (0.2)(-100) = 140/8 months$$

$$V_{2.1}(2) = 140 - 100 = 40/8 months$$

$$V_{2.2}(2) = E(B_{2.2}(2)) - C_{2.2}(2)$$

$$E(B_{2.2}(2)) = (0.5)(300) + (0.5)(-100) = 100/8 months$$

$$V_{2.2}(2) = 100 - 80 = 20/8 months$$

$$V_2(2) = \max\{40, 20\} = 40/8 months$$

We choose $A_{2.1}$ (licence).

Stage 1

$$V_{1.1}(1) = \max\{V_{1.1}(1), V_{1.2}(1), V_{1.3}(1)\}$$

$$V_{1.1}(1) = E(B_{1.1}(1)) - C_{1.1}(1)$$

$$E(B_{1.1}(1)) = (0.3)(300) + (0.7)(-20) = 76/10 months$$

$$V_{1.1}(1) = 76 - 80 = -4/10 months$$

$$V_{1.2}(1) = (0.5)(500) + (0.5)\left[-100 + 40\frac{1}{(1.1)^2}\right] - 400 = 216.5/10 month$$

$$V_{1.3}(1) = (0.8)\left[150 - \frac{25}{(1.1)^5}\right] + 0.2(100) - 100 = 27.6/10 months$$

$$V_{1.1}(1) = \max\{-4, -183.5, 27.6\} = 27.6$$

We choose $A_{1.3}$ (restructure on a smaller scale - a subset of programs). *Conclusion* At $t = 0$, we choose first to restructure on a smaller scale, and after 5 months we choose to switch to wholescale restructuring, the investment costs being \$100K in either case.

ii) Example 2: The value of waiting before deciding

A software developer **D** has a novel solution about Internet agents, and has found two stakeholders interested to finance this research. There are two possibilities:

- stakeholder Y is immediately interested to finance the research and offers $50,000
- stakeholder X doesn't have the possibility to evaluate it now but after 2 months and the potential available amounts for financing the design are: $100,000 for complete financing and $70,000 for partial financing.

The initial DT is presented in Fig.11.

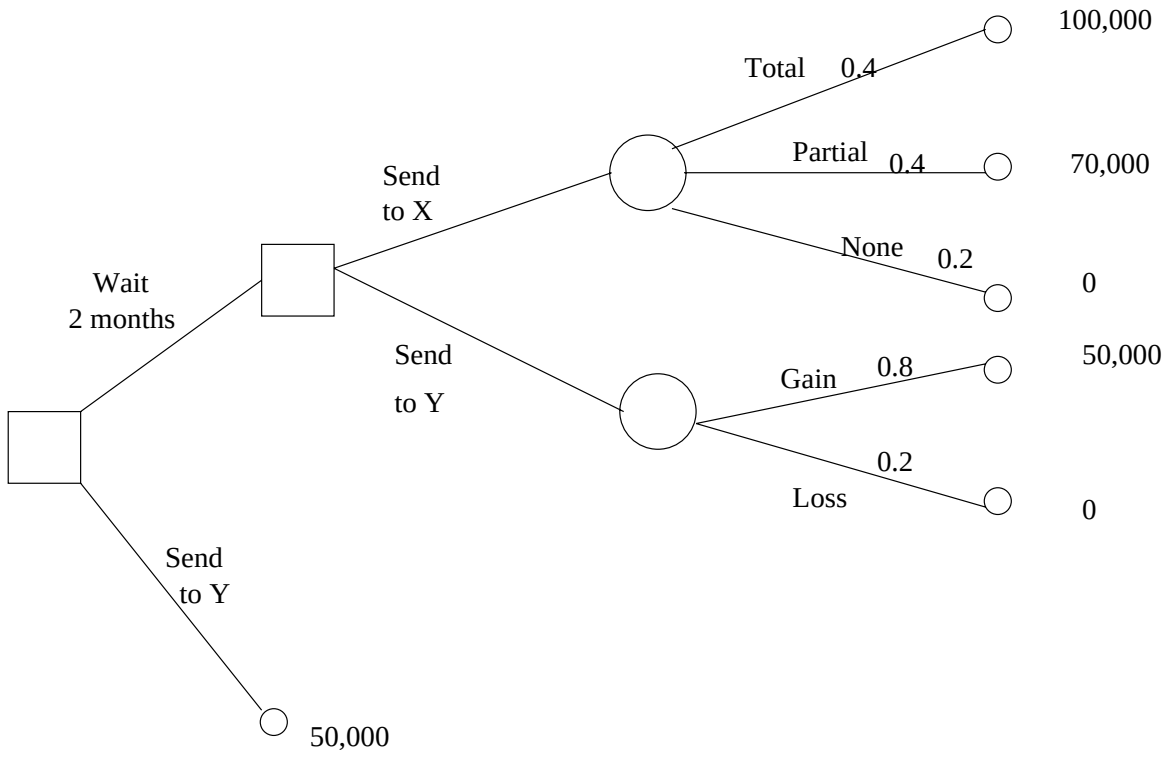

Wait
2 months
Send
to Y
50,000
Send
to X
Send
to Y
Total
0.4
100,000
Partial
0.4
70,000
None
0.2
0
Gain
0.8
50,000
0.2
Loss
0

Figure 11:

Waiting for 2 months offers the possibility for X to ask for an expert opinion about the technical compatibility between the offer and X's objectives. The expert answers are "favorable" or "unfavorable". There is also a tendency to answer "yes"even if the research cannot be financed, and this is captured by the conditional probabilities of the experiment associated to the proposal evaluation.
The information matrix is given in Tables 4 and 5.

Table 5:

Messages (r_j) States of the world(s_i)	$r_1 = favourable\ response$	$r_2 = unfavourable\ response$
$s_1 = X finances$	1	0
$s_2 = X partially$ finances	0.9	0.1
$s_3 = X doesn't$ finance	0.3	0.7

Table 6:

$P(r_1/s_1) = 1$	$P(r_2/s_1) = 0$
$P(r_1/s_2) = 0.9$	$P(r_2/s_2) = 0.1$
$P(r_1/s_3) = 0.3$	$P(r_2/s_3) = 0.7$

In Fig. 12 it is presented the DT associated to the proposal evaluation.

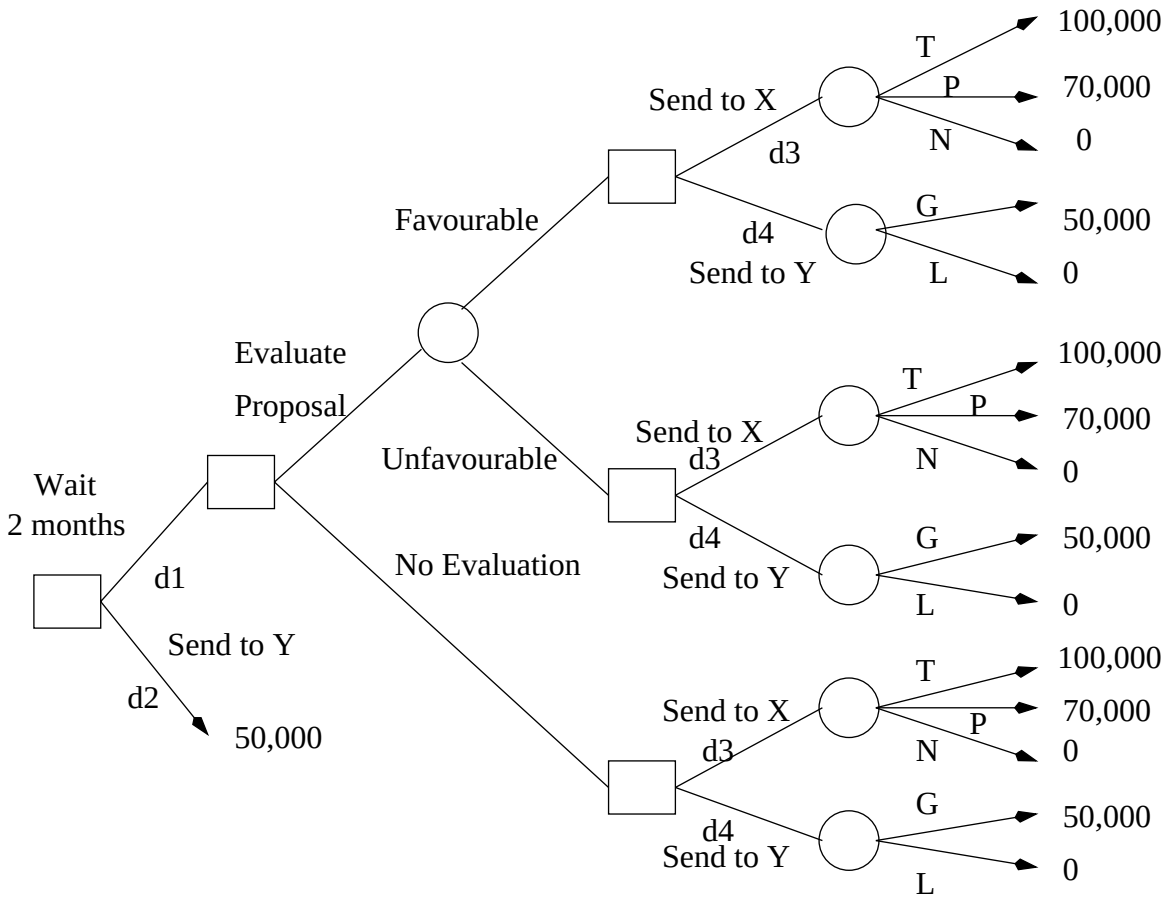


Figure 12:

The **decision problem** is:
i) Should **D** wait or not for 2 months in order to send his proposal to X or Y?
ii) After 2 months, which is the optimum path, taking into account that Y's offer is still available, but there are some modifications due to the time lag in submitting it?
Solution
The decision set $D = \{d_1, d_2, d_3, d_4\}$
The consequence set
$C = \{c_1, c_2, c_3, c_4\} = \{\$100,000; \$70,000; \$50,000; 0\}$
The states of the world set $S = \{s_1, s_2, s_3, s_4, s_5\}$:
s_1 - X offers total financing (T)
s_2 - X offers partial financing (P)
s_3 - X doesn't finance (N)
s_4 - Y finances (G)
s_5 - Y doesn't finance (L)
The experiment decision set $E = \{e_0, e_1\}$
e_0 - doesn't evaluate (no experimentation)
e_1 - evaluate proposal
The experiment result set $R = \{r_1, r_2\}$
r_1 - favourable
r_2 - unfavourable

From the information matrix (represented in Tables 4 and 5) we know the conditional probabilities $P(r_j/s_i)$ and from the decision tree shown in Fig.11 we know the following a priori probabilities:

$$P(s_1) = 0.4;\ P(s_2) = 0.4;\ P(s_3) = 0.2;$$

$$P(s_4) = 0.8;\ P(s_5) = 0.2.$$

Using these probabilities and the information matrix (Table 5) we compute:

$$
\begin{aligned}
P(r_1) &= P(s_1)P(r_1/s_1) + P(s_2)P(r_1/s_2) + P(s_3)P(r_1/s_3) \\
&= (0.4)(1) + (0.4)(0.9) + (0.2)(0.3) \\
&= 0.4 + 0.36 + 0.06 = 0.82
\end{aligned}
$$

$$
\begin{aligned}
P(r_2) &= 1 - P(r_1) = 0.18 \\
P(s_1/r_1) &= \frac{P(s_1)P(r_1/s_1)}{P(r_1)} = \frac{(0.4)(1)}{0.82} = 0.49 \\
P(s_2/r_1) &= \frac{P(s_2)P(r_1/s_2)}{P(r_1)} = \frac{(0.4)(0.9)}{0.82} = 0.44 \\
P(s_3/r_1) &= \frac{P(s_3)P(r_1/s_3)}{P(r_1)} = \frac{(0.2)(0.3)}{0.82} = 0.07 \\
P(s_1/r_2) &= \frac{P(s_1)P(r_2/s_1)}{P(r_2)} = \frac{(0.4)(0)}{0.18} = 0.0 \\
P(s_2/r_2) &= \frac{P(s_2)P(r_2/s_2)}{P(r_2)} = \frac{(0.4)(0.1)}{0.18} = 0.22 \\
P(s_3/r_2) &= \frac{P(s_3)P(r_2/s_3)}{P(r_2)} = \frac{(0.2)(0.7)}{0.18} = 0.78
\end{aligned}
$$

The DT in Fig.13 contains now the conditional probabilities previously determined:

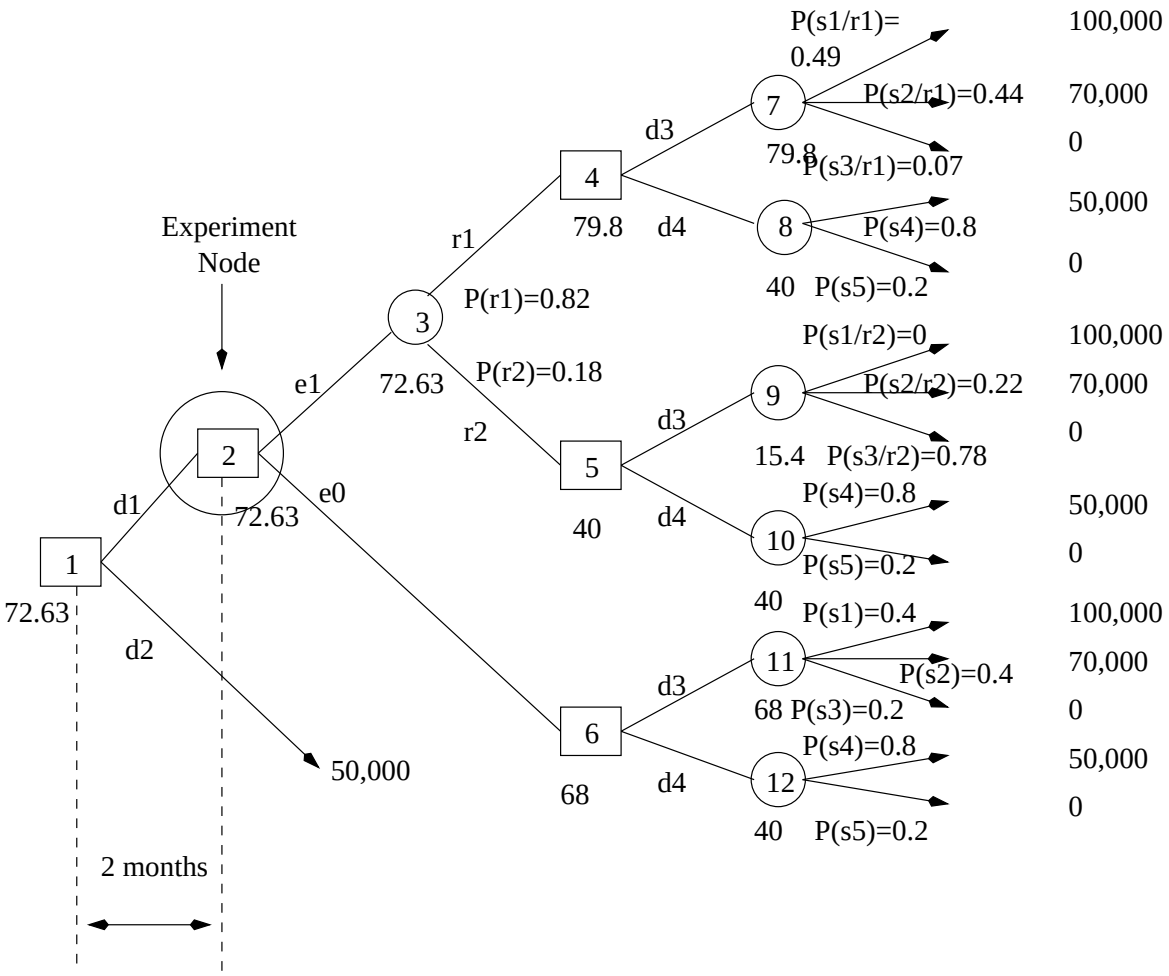


Figure 13: DT with conditional probabilities.

Using the information specified in the DT from Fig.13 we can compute the expected value of each event node and the optimal decision to be made in each decisional node using the dynamic programming approach:

$$E_7 = P(s_1/r_1)(100) + P(s_2/r_1)(70) + P(s_3/r_1)(0)$$

$$= \quad 49 + 30.8 = 79.8$$

$$E_8 = P(s_4)(50) + P(s_5)(0) = (0.8)(50) + 0 = 40$$

$$E_4 = \max\{E_7, E_8\} = 79.8 \Rightarrow d_3$$

$$E_9 = P(s_1/r_2)(100) + P(s_2/r_2)(70) + P(s_3/r_2)(0)$$

$$= \quad (0)(100) + (0.22)(70) + 0 = 15.4$$

$$E_{10} = P(s_4)(50) + P(s_5)(0) = (0.8)(50) + (0.2)(0) = 40$$

$$E_5 = \max\{E_9, E_{10}\} = 40 \Rightarrow d_4$$

$$E_{11} = P(0.4)(100) + (0.4)(70) + (0.2)(0) = 68$$

$$E_{12} = (0.8)(50) + (0.2)(0) = 40$$

$$E_6 = \max\{E_{11}, E_{12}\} = 68 \Rightarrow d_3$$

$$E_3 = P(r_1)(79.8) + P(r_2)(40) = (0.82)(79.8)$$

$$+ \quad (0.18)(40) = 72.63$$

$$E_2 = \max\{E_3, E_6\} = 72.63 \Rightarrow e_1$$

$$E_1 = \max\{E_2, 50\} = 72.63 \Rightarrow d_1$$

The **conclusion** which results from our analysis and theoretical framework is:
The initial optimal decision (at t=0) is to wait for 2 months, after which an experiment is performed (an expert evaluation for X). If the answer is favourable (the probability being equal to 0.82), then send design solution to X because the payoff is higher ($79.8K versus $40K). If the answer is unfavourable (the probability being equal to 0.18), then send the design proposal to Y because the payoff is higher than the payoff obtained by sending it to X: $40K versus $15.4K.

6. Final Remarks

- The paper defines and presents the application of the model system concept, connected with the critical success factors in software projects and the current and future trends in software system development and evolution.
- Model systems allow to:
  - i)represent their multiple facets
  - ii)analyse their integration
  - iii)enhance their capabilities in different areas.

DF is a formal framework for assisting optimal dynamic decision-making in the software process based on an economic criterion

- combines synthetic, inductive, with analytic, deductive approaches
- can be applied in decisional situations referring to
  - product structure(such as: COTS, reusable components, architecture, reengineering)
  - process structure (e.g. stages)
  - timing of commitments to various actions.

The framework is based on a computational model for analysing alternatives, states of the world, risks, options to wait, benefits, costs formulated in terms of

- time value of money
- information value to support decisions

in view of dynamic optimization of their recommended sequence over a defined time horizon. DF is useful in an iterative software development where the risk-driven iterations are defined as sequences of decisions about activities with an established plan and evaluation criteria, resulting in an executable release.