Chapter 2

A Conceptual Model of Software Development

> "The purpose of science is not to analyze or describe but to make useful models of the world. A model is useful if it allows us to get use out of it."
> Edward de Bono

To explore the many facets of *software project management*, we introduce a conceptual model of software development. This model (or ontology) of software projects is organized around eight key concepts and their relationships that are universal across all software projects:

Intent, Product, Work, People, Time, Quality, Risk & uncertainty, Cost and Value,

are found in some ways or another in every software development project, whatever size, genre, type or color.

Outline

- Four core entities
  - Intent - Product - Work - People
- Three fundamental attributes
  - Time - Quality - Risk
- Project, defined
- Adding the concepts of Cost and Value
- Mapping of onto common project artifacts
  - Intent - Product - Work - People

Four core entities

There are four core entities in our model of software development: Intent, Product, Work and People; see figure 1.

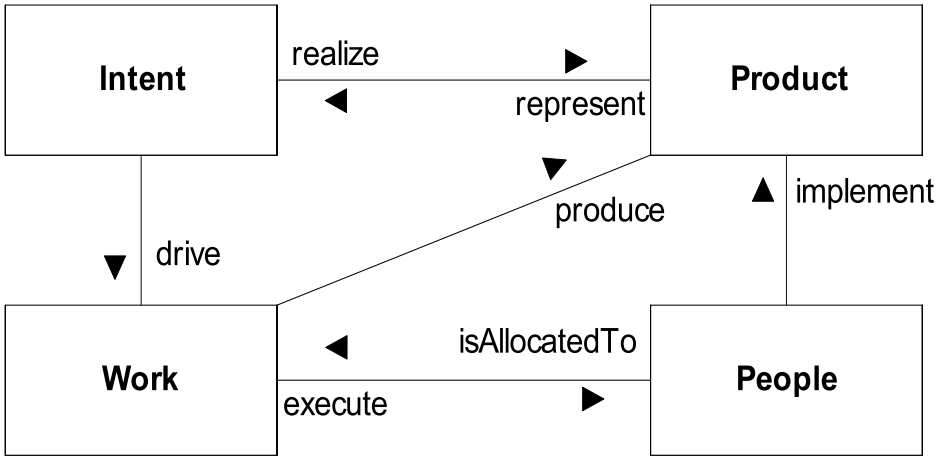

Figure 1: Four core concepts in software development: Intent, Work, People and Product

Intent

The concept of *Intent* denotes what the project is trying to achieve. The *Intent* defines the scope of the project, the intentions and hopes of the key stakeholders, the objectives. While we think of the intent as "the requirements" or "the specification", in practice *Intent* may take many diverse forms: a set of tests that the product must pass contributes to define *Intent*. A set of software problem reports that must be dealt with also indirectly defines *Intent*. Various constraints, implicit or explicit, internal or external to the project will also affect *Intent*. And one constant of software projects is that they are under pressure of a stream of change requests which modify the *Intent*.

Product

The concept of *Product* denotes the outcome of the project, what has been achieved. This is the actual software, accompanied with any other artifacts that are needed to make it a complete product: an installer, a set of data, the user's guide, some training material, etc. Why aren't *Intent* and *Product* more or less equivalent? Why do we need to distinguish them in our model? *Intent* precedes *Product*: *Intent* is an abstraction, a virtuality that sketches the reality that the project is set to achieve. Even when the product is "done" there may be discrepancies between the *Intent* and the *Product*; the *Intent* may have evolved in the meantime, or the *Product* has come short in some ways of the original Intent. These discrepancies between *Intent* and *Product* are the key drivers for the project; they are *the imbalance that makes it run*. Imagine the relationship between *Intent* and *Product* as a bungee cord: the further apart and the more energy the project will expend to bring them closer.

Work

The concept of *Work* denotes the activities, tasks, steps that need to be accomplished in order to turn *Intent* into *Product*. They often come defined by a process, or a method, which attempts to describe a systematic way to build a product; some elements of Work are defined "on-the-fly" in an ad hoc fashion. In many cases, a Work item produces or refines some artifact: a document, a model, an idea, a piece of code, a report, formal or

informal. Some of these artifacts are only useful internally to the project, as stepping-stones, and do not appear in any form in the final product. They are not “deliverables”.

People
The concept of *People* is important in modern software project management because they are the main “engine” behind *Work* elements. Software development is an intellectual activity that is very ‘human-intensive’. Most of the work elements are done by human beings, and only little of this work can be automated. So the availability and the competence of the people are keys to get all the work done. Also most of the cost of software development is associated with people. (We will often use the word Staff and use the initial S to denote the concept of *People* and not clash with the P of product).

Three fundamental attributes

Each of these four core concepts has 3 attributes: Time, Quality and Risk.

Time
The concept of *Time* is orthogonal to our fundamental quadruple [Intent, Work, People, Product]. Often we will use the phrase *lifecycle* to denote what happens with a project over time. It is tempting to define a project linearly relative to time in 5 main steps: 1) define completely the Intent, 2) derive from the Intent all the Work that needs to be accomplished, 3) allocate work to People, and 4) People build the Product, 5) which acceptance testing will show that it matches exactly the original *Intent*. This has been tried again and again, but with very meager successes in software development for a range of reasons that we will examine later on (chapter 5 and 6). In reality, we define *Intent* gradually, and it tends to evolve throughout the project under various pressures and demands for changes. We can therefore only define part of the *Work* at any point in time, and allocate it to *People*, who will therefore only build part of a *Product*. This partial product will influence back the Intent, through user feedback, or problem reports. It will also influence how people will conduct the work in the future. Other chunks of Intent are then carved out, more *Work* defined, and the *Product* will evolve until it reaches a deliverable stage. All modern software development approaches are iterative and incremental, and they define a project as a sequence over time:

$$\{ [Intent_1, Work_1, People_1, Product_1], \\ [Intent_2, Work_2, People_2, Product_2], \\ \ldots \\ [Intent_n, Work_n, People_n, Product_n] \ \}$$

where $Product_n$ is the final ‘deliverable’.

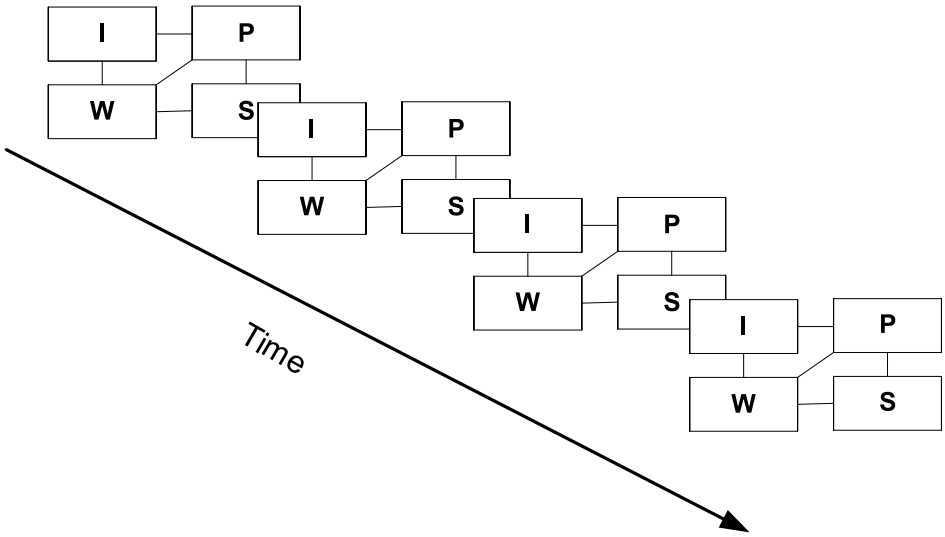

Figure 2: Intent, Work, People and Product evolve over Time

Quality

The concept of *Quality* is also an orthogonal notion to our fundamental quadruple [Intent, Work, People, Product]. We can see quality as an attribute of each of them. Quality of the Intent denotes how good we are at defining and planning a Product. Quality of the Work denotes the quality of the process we use to develop software and all the intermediate artifacts. Quality of the People denotes the competence and diligence and dedication of the staff assigned to the project, and finally quality of the product is a measure of how close to the expectation of the stakeholders the delivered product is. These four aspects of quality may evolve over time, following the sequence we described above, and hopefully their quality increases over time (inasmuch as quality is quantifiable).

Risk and uncertainty

Finally the concept of *Risk* denotes the uncertainty that is associated to each of the four fundamental concepts at some point in time: uncertainty in the Intent, because the domain is new, for example, uncertainty in the Work to be performed, because the process is unclear, risks associated with the people and therefore uncertainties in the final Product. Similarly to Quality above, these uncertainties evolve over time: the risks will be mitigated, unknowns will become known, but new risks keep emerging.

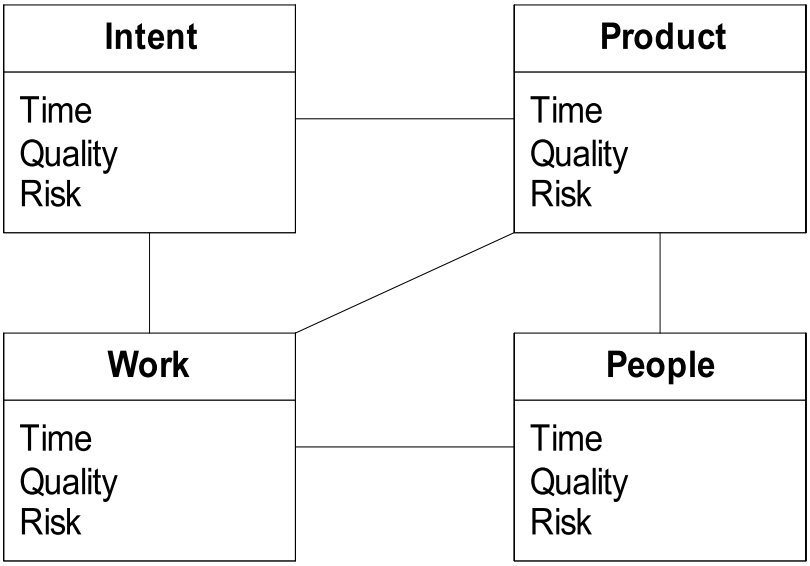

Figure 3: Time, Quality and Risk are attributes of Intent, Work, People and Product

Value and Cost

Finally, *Value* is associated with Intent and Product: we need to assign expected value to the Intent to guide development of the Product over time, while the *Cost* of the development is associated with the Work and the People. As software is essentially an intellectual, human-intensive activity, they both are directly derived of the cost of the People associated with the project and the Work they do: what, how much, for how long. This is a characteristic of software development not shared with other engineering disciplines, such as civil engineering.

Altogether a software development *Project* is all the *work* that people have to accomplished over time to realize in a *product* some specific *intent*, at some level of *quality*, delivering *value* to the business at a given *cost*.

The project and its context

A *software project* is temporary endeavour intended to create a new software product or service, or the software part of a software-intensive system. It is temporary in the sense that it has a definite beginning and a definite end, in contrast with a continuous endeavour, such as running the IT operations of an organization. A software project has specific and sometimes conflicting objectives and many constraints of diverse nature, mainly technical, temporal and financial. Software project management is therefore the art of balancing competing objectives, managing risks, and overcoming constraints to successfully deliver a product which meets the needs of both customers and users (the customer paying the bill not always being the end-user).

The software project, represented in our conceptual model with a tuple or composite object [Intent, Work, People, Product], or more precisely with a sequence of such composite objects, does not live in isolation, but it sits in a wider context, which is crucial to understand for a software project manager.

Intent and Product are mostly facing the users and customers community, intent driven mostly by them, and the product delivered to them. Constraints come from the customers, and from the business environment: the company which “owns” the project. There are also constraints coming from legal and regulatory bodies in some industries, especially in the safety-critical domains: transportation, defense, biomedical, nuclear, and in the financial domain.

People and Work are mostly influenced by the available technologies to develop and deploy the software product or service: programming languages, methods, development environment, software tools, reusable components, deployment platforms: CPU, OS, network protocols, etc.

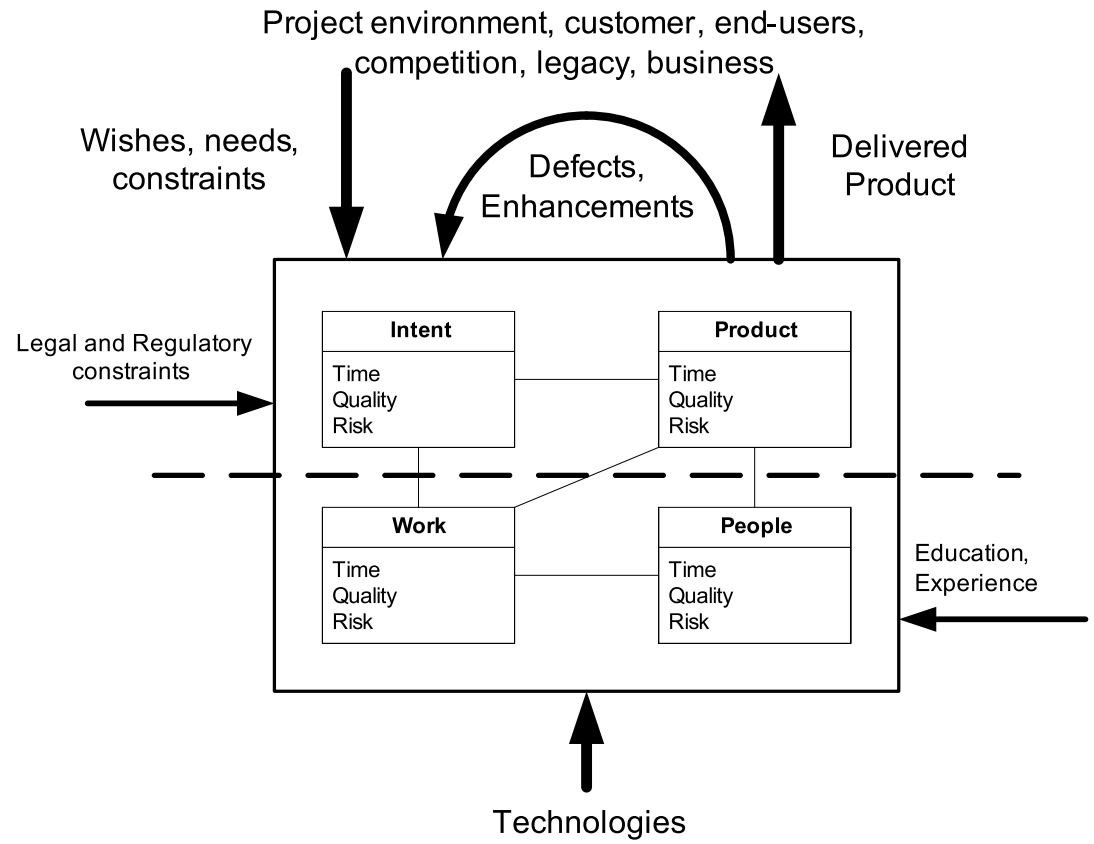

Figure 4: The project and its context

Mapping the core entities

Intent, revisited

The Intent of the project is an image, a description, a model of what the various parties involved want to product to be. This Intent may take several forms, depending on the type of software project and on the method or process used. We will assume that the Intent can be decomposed in a set of Intent elements, coming from various sources, and carrying different names in different methods:

- *Users needs*: a description of the needs of the user, at least the needs that we intend to satisfy
- *Vision*: a document that describes in high level terms (RUP)
- An initial product *backlog* (Scrum)
- A Software Requirement Specification, SRS (IEEE Standard 830)
- A list of user stories (XP)
- A feature list (FDD)
- A use-case model, with or without supplementary specifications for non functional requirements (RUP)
- A set of acceptance test cases (TDD)
- A list of software problem reports (or bugs, or defects)
- A prototype or mock-up
- An existing product (if we are migrating or re-engineering a legacy application, for example)

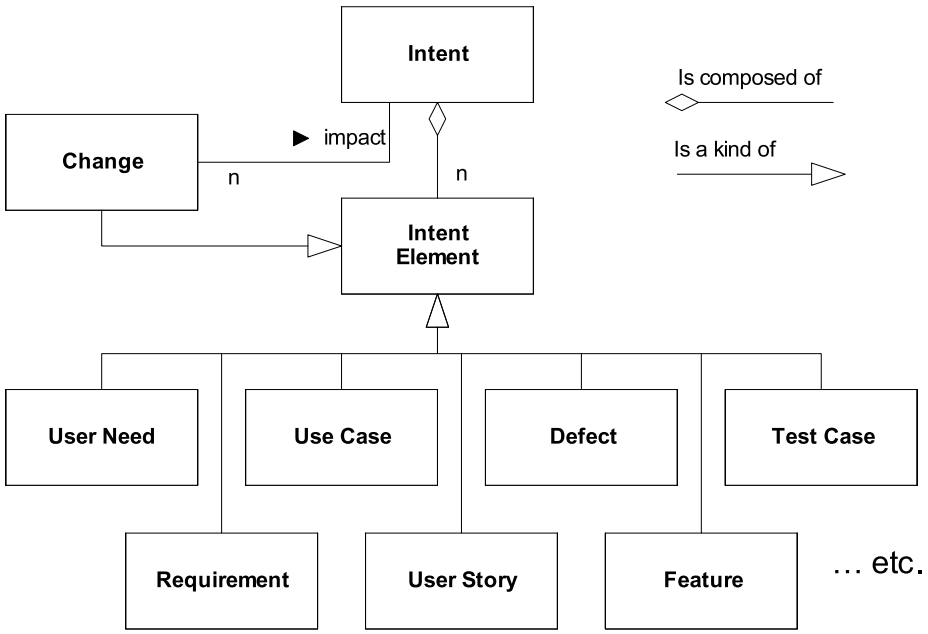

Figure 5: Intent elements come in many different forms, and Intent constantly changes

It's around the concept of *Intent* that we can introduce the subtle but pervasive concept of *change*, since changes occurring along time in a project are in most cases changes in Intent, which in turn will trigger changes in work and changes in the product. (There maybe also changes in people, though, not related to intent: a resignation, for example.)

Work, revisited

Work is the stuff that makes traditional schedules and plans with associate with project management: work items are found in network schedules, Gantt charts, in tools such as Microsoft Project®, Niku® or Primavera®. Large chunk of work constitute Work Breakdown Schedules (WBS), used for planning projects. Smaller work items are the items that developers put on their to-do list, scribble on their white boards or their PDAs.

The bulk of process descriptions, such as RUP® (IBM, 2007) or MSF® , are

dedicated to the description of work: sub-processes, activities, tasks, steps. They are the elements of focus in software process standards such as IEEE-1074 (1997) or ISO 12207 (1995)

One of the most difficult tasks of software project management is to derive from a given *Intent* the required *Work*. We still only know how to do this very approximately, and there are many work items that spring out spontaneously during the course of a project, due to unknowns, to people making errors and other various mishaps.

The amount of Work is also a key ingredient to the estimation of effort and schedule, and therefore to the cost of the project. As we will see later, effort estimation is another big hurdle in software development, the “black art”. Finally, project management attempts to monitor progress while the project is on-going by comparing actual work performed to the anticipated work.

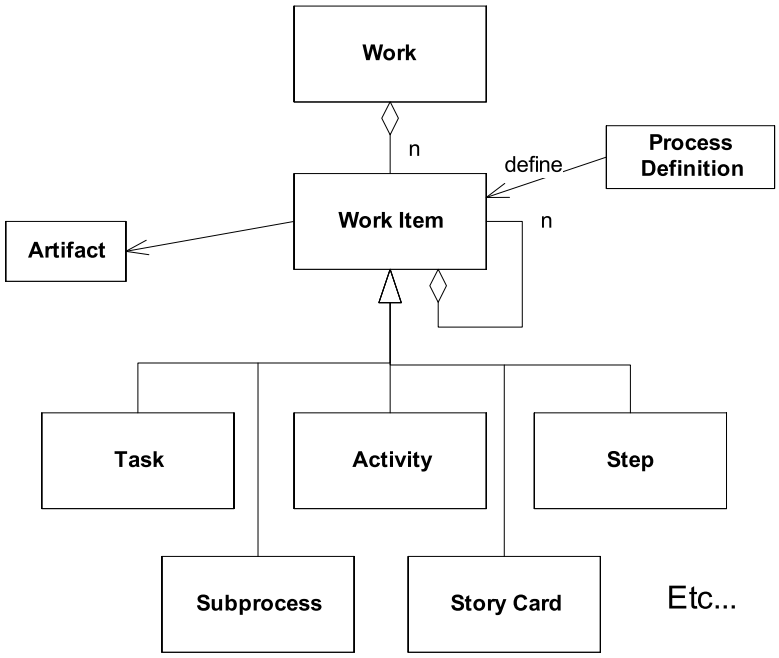

Figure 6: Work Items come in many different forms and size

Many work items ‘operate’ on some artifact, that is, they use artifacts as input and create or update artifacts. These artifacts, their templates, and the details of the work are part of the project’s process, and these templates can evolve over time during a project.

Product, revisited

One could claim that the delivered product is in the end the only thing which should matter to the software project manager. What constitutes the product will vary greatly across domains, from tiny software embedded in some device, to large distributed systems, upgraded dynamically weekly, from shrink-wrapped software sold over the internet to “software as a service”, from one-off “Kleenex” software rapidly assembled out of a software junkyard to mission-critical applications maintained over 25 years. In the simple cases, the product consists in executable code, often targeted to a specific

set of operating systems, accompanied by application data, and some supporting material: user guides, training material, etc. Nowadays products are often expected to run on multiple OS platforms, and several versions of these, as well as supporting different locales: languages and work habits specific to the countries of the user.

A notable concept associated to the product is that of a *release*, which is a product made available to certain parties at given points in time during the project.

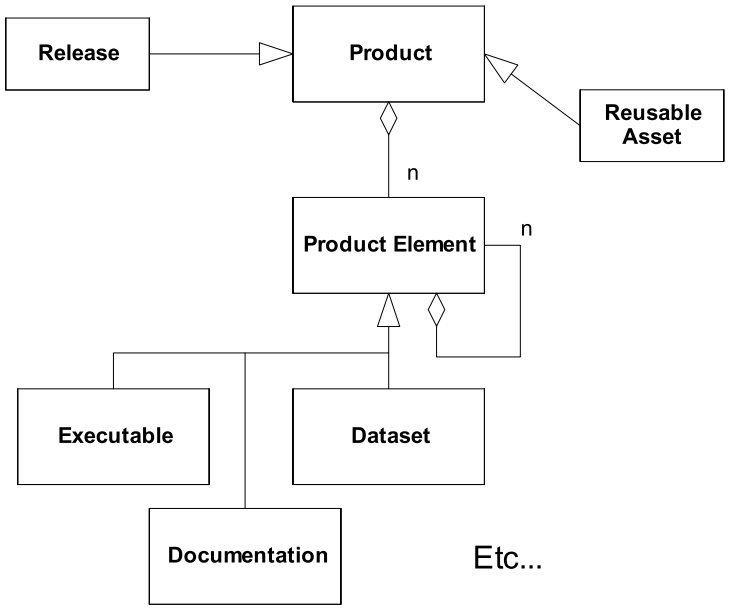

Figure 7: Elements of a Product vary across types of projects; a Release and Reusable assets are distinguished forms of Product

It is around the concept of product that we can discuss issues such as software as an asset, intellectual property rights, and the reuse of software assets from project to projects, whether this software is open-source, commercial-off-the-shelf, or proprietary.

People, revisited

We've come to realize, through some pains, that software engineering is not primarily a technical issue, but a people issue. Most of the real difficulties in software development, most of the errors and shortcomings are not related to technologies but to the people developing it, their competence, experience and availability, the communication and coordination between these people or teams of people. Therefore the staff component, which is very often ignored in process standards and methods, or abstracted as some kind of vague and perfect agent, plays a major role in our conceptual model. There are several aspects crucial to software project management: the persons themselves, i.e., the individuals, with their knowledge and competence, the roles they play in the software development process: analyst, developer, tester; their organizations in teams, and the allocation of work to people or teams.

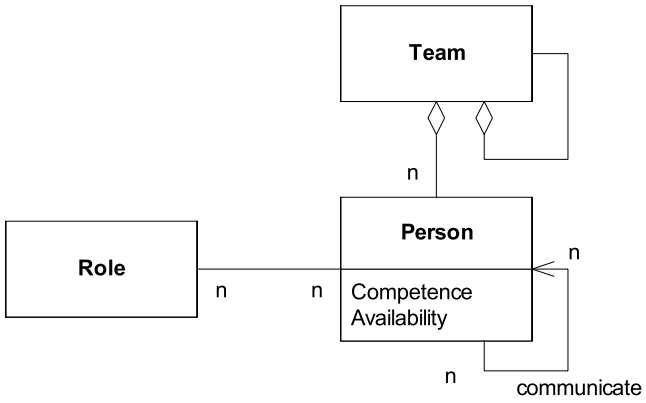

Figure 8: Persons, teams and roles

It's also around our concept of people that we can discuss issues such as ethics and professional practice.

Summary

All software development projects can be described as endeavours aiming at bridging the gap between some *Intent* (a vision, a plan), and a *Product* (a tangible reality). Many of the issues behind software project management reside in managing the various concepts presented, especially the hidden ones: Work, People, Time, Quality and Risk, and understanding their relationships.

We now have with this model a basis for describing and discussing many aspects of software project management. Chapter 5 will dive into risk and uncertainty, chapter 6 and 7 into time and work, chapter 8 into quality. Intent will be at the center of chapter 9, people will be discussed in chapters 14, 15 and 16.

Further reading

For a good overview of the issues behind software development, you must read the all time classic by Fred Brooks: *No Silver Bullet* (Brooks, 1986), and the reply by Dan Berry (2002) on the *Pain of Software Development* [note: add URLs to biblio]

Joel Jeffrey defined a more elaborate model of software development, when he was trying to get a the core of the issues in software engineering (Jeffrey, 1996), and he has inspired in part the model presented here.

The Function-Behaviour-Structure (FBS) framework of engineering design in general developed by John Gero and Udo Kannengiesser (Gero, 1990; Gero & Kannengiesser, 2004), and which we mapped to software (Kruchten, 2005) has also helped shaped this model.

Bibliography

Berry, D. (2002). *The Inevitable Pain of Software Development: Why There Is No Silver Bullet.* Paper presented at the Workshop on Radical Innovations of Software and Systems Engineering in the Future, Monterey, CA.

Brooks, F. P. (1986). No Silver Bullet-Essence and Accident in Software Engineering. In H.-J. Kugler (Ed.), *Proceedings of the IFIP Tenth World Computing Conference* (pp. 1069-1076). Amsterdam, NL: Elsevier Science B.V.
Gero, J. S. (1990). Design prototypes: A knowledge representation scheme for design. *AI Magazine, 11*(4), 26-36.
Gero, J. S., & Kannengiesser, U. (2004). The situated function–behaviour–structure framework. *Design Studies, 25*(4), 373-391.
IBM. (2007). Rational Unified Process. Retrieved from http://www-306.ibm.com/software/awdtools/rup/?S_TACT=105AGY59&S_CMP=WIKI&ca=dtl-08rupsite
IEEE. (1997). *Standard for developing software life cycle processes*. New York: IEEE Standards Association.
ISO/IEC 12207. (1995). Information technology - Software life-cycle processes. Geneva: ISO.
Jeffrey, H. J. (1996). Addressing the essential difficulties of software engineering. *Journal of Systems and Software, 32*(2), 157-179.
Kruchten, P. (2005). Casting Software Design in the Function-Behavior-Structure (FBS) Framework. *IEEE Software, 22*(2), 52-58.