Towards a method for improving globally distributed software development using quantitatively measurable process patterns

The Problem-Goal-Pattern-Measurement approach

M.M. Walgers
m.m.walgers@student.utwente.nl

ABSTRACT

Globally Distributed Software Development is being used more and more in software development organizations as it offers great advantages. However, GDSD brings its problems regarding communication, coordination, and control as well. Knowledge of how to solve or ease these recurring problems is existent, but rarely documented in a reusable pattern format. This paper proposes a method for documenting and structuring the knowledge body of the solutions to these recurring problems. The importance of quantitative measurement of solutions is elaborated, and a pattern language for how to document measurable solutions as patterns is defined. Some example quantitatively measurable process patterns are documented using this defined pattern language. The process of how patterns and their quantitative measurements should be used to improve the software development process is also described.

Keywords


1. INTRODUCTION

More and more the development of software is distributed across different locations. Global software development is becoming a norm in the software industry. [DM06]. Businesses distribute their software production globally because of economic, political and practical needs. [MW01] It could be to reduce development cost, reduce time-to-market of a product, a broader employee pool, a closer relation to the area of deployment of the product or many more reasons. [DM06] [CA01] A combination of these factors are also very common. The body of knowledge regarding Global Software Development is still increasing as the art and science of organizing and managing globally distributed software development is still evolving. [DM06]

The globally distributed development of software brings some difficulties as well, especially regarding communication, coordination and control. [MH01] [HM01] [CA01] [MW01] [LLL06] As these problems occur across all Globally Distributed Software Development (GDSD) projects there are many documented solutions to ease or solve these problems. If the same general solution is used more often in similar situations to solve similar problems then the solution can be documented as a pattern. A pattern is a general solution abstracted from documentation of best practices which can be applied as a general solution to a generic problem. The results of these patterns are not always (quantitatively) measurable.

However, the patterns for which the results are measurable can for example be implemented in a tool which can point out areas of the project that can be improved by the use of a certain pattern. By using quantitative measures one can assess the current situation, locate possible areas of improvement and predict the possible improvements of a solution. While there are a number of suggestions on how to improve globally distributed software development, there is no research investigating their effectiveness. [MH01] And as the importance and complexity of GDSD increases the use of effective strategies and practices to successfully organize and manage global software development becomes critical. [DM06] Effectively and efficiently managing GDSD is not easy and if done badly can even risk the overall productivity [MH01] [EN01] [MW01] [HM01] [ORM04]. As the complexity of the product increases so does the complexity of the project management. So project management requires a structural approach to handling the problems of GDSD. [DM06] For this purpose the manager needs the ability to assess the current situation, reason about the effects of certain solutions and predict the future. A good basis for assessing and reasoning about situations and effects of solutions is measurement, as the corollary rule of the rule defined by Tom DeMarco cites: “you can neither predict nor control what you cannot measure”. [FP97]

This paper first will first elaborate on the advantages and problems of globally distributed software development. Then it will take a closer look at (process) patterns and the quantitative measurability of patterns. Then this paper will present a method of how to structure and document the (available) knowledge body of (quantitatively measurable) patterns. The process of how patterns and their quantitative measurements should be used to improve the software development process is described. And finally some (example) patterns are documented using the defined pattern language for quantitatively measurable (process) patterns as well.

2. WHY USE GLOBALLY DISTRIBUTED SOFTWARE DEVELOPMENT

Software products are becoming increasingly large and complex. The development of these large software products, often indicating large development projects, requires much effort. But as the software becomes more complex, the demands on the development company are becoming more rigorous as
well. The quality of the delivered products has to increase, but at the same time the development cost as well as the development time has to decrease. By using globally distributed software development the companies try to react to these demands. The potential of globally distributed software development is high. However, globally distributed software development projects are not easy to manage and if done badly can even result in a lower success of projects then when they would have been developed locally. [MH01] [EN01] [MW01] [HM01] [ORM04] The benefits can be huge, but one has to be careful of the pitfalls. Globally distributed software development has its possible advantages, but brings its own problems as well.

Besides the lower cost of development in developing countries such as India, Russia, China, and Bulgaria [Sak04] (due to the fact that the wages of employees there are lower then those of comparable employees in for example the Netherlands) there are other benefits to the company using globally distributed software development as well. Not only is the employee pool larger, the (area specific) experience of employees in another country can be larger as well. The company can use the distributed sites to have developers closer to the client, who can help in understanding client requirements and therefore increasing client satisfaction with the eventual product as the product is being (co-)developed by a team with local knowledge. The use of globally distributed sites for software development can decrease the time-to-market of a product as well by exploiting the time differences between the sites. By using well located different sites round-the-clock development can be achieved; which is clearly desirable if wanting to reduce the time-to-market. [MH01] [DM06] [CA01]

3. WHAT ARE PROCESS PATTERNS
In the mid-1970’s Christopher Alexander introduced the idea of formulating rules of thumb in architectural design into patterns. The solutions to commonly encountered problems were recorded in a reusable form to share the knowledge of proven solutions. [Ale79] [Har05]. In 1995 Erich Gamma et al. introduced the design patterns for software. [GHJV95] Process patterns were first introduced by James Coplien in 1994 at the PLoP’94 conference and published in 1995. [CS95]

Different globally distributed projects often face the same (recurring) problems. [HM01] Often these problems are solved in a similar way at the different projects. [HM01] If the solution to the problem seems to be successful and the same solution is being applied more often, possibly in different projects by different companies; then it will be regarded as ‘best practice’. If company or situation specific aspects are abstracted from the solution, then the solution can be used as a general solution to the problem and can be documented as a solution pattern. There are many different kinds of patterns like design patterns [GHJV95], analysis patterns [Fow97], organizational patterns [CH04] and process patterns [CS95], but in this paper they are simply referred to as patterns as the context makes clear it is about process patterns.

As described in [GHJV95]: a pattern is a documented (proven) solution to a recurring problem which contains:

- the name of the pattern
- a description of the recurring problem
- the core of the solution to that problem
- the consequences of that solution.

Lasse Harjumaa describes patterns as ‘concrete and manageable descriptions of improvement actions needed in various situations’. [Har05]

There are various suggestions of structuring patterns in the literature as for example in [Har05], but they all contain at least the information described above.

Even though patterns are most widely utilized for software architecture and design, using patterns for software development processes and organizations with patterns has become increasingly popular. [Har05]

A process pattern is a solution to a problem (occurring during software development) that can be applied to the development process. Generally multiple patterns can be used at the same time to address multiple goals simultaneously. An easily understood example of a process pattern to solve communication problems is to organize regular meetings. Regular meetings between employees working within a project will increase the communication between the employees and therefore reduce the communication problems. [PL04] [RLKP06] Another example is the productivity and job satisfaction of employees which seems to be much higher in collocated teams then in distributed teams [CP06]. The development tasks of distributed projects seem to take longer then that of comparable collocated projects. [HM01] [HM01] So the solution pattern of collocating employees working on the same project as much as possible increases productivity and job satisfaction [CP06] [EN01] (although one can reason only the productivity is quantitatively measurable). Another pattern example of how to improve the GDSD is the use of chunking as described in [EN01]. Some specific process patterns will be described in more detail later on in this paper.

4. WHY QUANTITATIVELY MEASURABLE PATTERNS
The term ‘measurable’ implies that measurements can be applied to the patterns. The used definition of measurement is the following: “Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.” as stated in the book of Fenton and Pfleeger. [FP97, p.5]

As described in [FP97, p.88] ‘measurement allows managers and developers to monitor the effects of activities and changes on all aspects of development, so that action can be taken as early as possible to control the final outcome should actual measurements differ significantly from plans’. These activities could for example be the application of patterns. So measurement is important for three basic activities:

- to understand/assess what is currently happening; establish a baseline and be able to set goals for the future
- to control what is happening, control whether targets/goals are being met and predict what will happen in the future
- to improve the situation by assessing the measurements. [FP97, p.13]

Lasse Harjumaa even points out that patterns can be used to run preliminary trials of an improvement on smaller scale to check its success. [Har05].

As the corollary rule of the rule defined by Tom DeMarco, a strong supporter of the need for measurement in software development, mentioned in [FP97] asserts: “you can neither predict nor control what you cannot measure”. [FP97, p.14]
To know what aspects of the software development process to improve one first has to assess the current situation. By measuring and assessing the current situation with those measures one can point out specific areas where improvement could be necessary. By knowing the effects of certain solutions one can predict the situation after implementing the solution and assess whether it should be implemented or not; whether the effects of the solution are worth implementing it. Besides predicting the effects of implementing a solution, one can also use the measurability of the patterns to measure the situation in two different projects and compare the measures with each other. If there are differences one can look for, or study, the basis for these differences to improve the understanding of the solution in projects. However, it is not all that easy because the measurement of processes can be difficult. As the book of Fenton and Pfleeger notes: ‘process measures are often more difficult to define than product and resource measures, in large part because the process activities are less understood’. [FP97, p.37]

One must keep in mind that quantitatively measurable process patterns are by no means a silver bullet but merely a way to help improve the software development process.

5. DATA COLLECTION, DATA REPRESENTATION AND PROCESS MATURITY

For measurements one needs data. The quality of the measurement is clearly dependent on the quality of the collected data. [FP97] The collected data can address various aspects of the software development process and can be represented in different ways. A few important aspects for the software development process to acquire data from are information on the structure of the organization of the company (for example the employee structure and their relationships), the processes used by the company and their relationships, the planning of the project (for example the time schedule and the defined deadlines), the structure of the software being designed or developed, or a graph/table representing the interdependencies between the modules being developed, and of course any data of activities occurring during the project (for example productivity of employees or time spent on certain activities).

This data can be represented in various ways to allow measurement. For example in organizational diagrams, process diagrams or time-schedules.

To enable the gathering of quality data there is need for a certain process maturity. The Software Engineering Institute defined several levels (level 1 to 5) of software maturity, each one the foundation for the next. Ordered in increasing maturity order these are: initial (ad hoc), repeatable, defined, managed and optimizing. These levels are defined in the Capability Maturity Model [PCCW93], also known as CMM, and are therefore often referred to as CMM maturity levels or simply CMM-levels.

Process maturity positively impacts global development. [EN01] At a given maturity level you can collect the measurements for that level and all levels below it. Sometimes some measures of a higher level can be done as well because a project can be more mature in some parts than in others. By collecting these measurements one can help in increasing the maturity of the entire project. [FP97, p.93] Each level of process maturity has its own set of measurable data. So the more mature a project is, the more data can be acquired and therefore measured. Ebert and De Neve state that a CMM maturity level of at least 2 is an absolute minimum to achieve sound project management (which is a prerequisite for distributed development). [EN01] So quantitatively measuring processes of a project at level 1 is impossible or at least not very useful. In fact it is stated in the CMM standard [PCCW93] that quantitative process management, which is process management based on quantitative process measurements, is not a key process area until a project reaches level 4 because only then does the project understand the processes well enough and are the processes well enough visible to allow quantitative measurement and apply quantitative process management.

6. PROBLEMS OF GLOBALLY DISTRIBUTED SOFTWARE DEVELOPMENT

Although globally distributed software development creates many opportunities it creates many more problem areas as well. These potential problem areas can turn into problems if they are not addressed properly. To address these problem areas there are many solution patterns; a few of which are addressed later on in this paper.

One of the characteristics of the problems of globally distributed software development is that the problems are often project independent, meaning they occur across all projects.
There is a lot of literature on, for example case-studies all listing the same general problems. [MH01] [HM01] [CA01] [MW01] [LLL06] If categorized all of these listed problems can be addressed under communication, coordination or control problems. For the sake of differentiating between communication and coordination we, instead of using the Malone and Crowston model [MC94], use the definitions of communication, coordination and control as defined in [CA01].

‘Coordination is the act of integrating each task with each organizational unit, so the unit contributes to the overall objective.’

‘Control is the process of adhering to goals, policies, standards, or quality levels.’

‘Communication is a mediating factor affecting both coordination and control.’ [CA01]

Figure 2. Effects between distance, communication, coordination and control [CA01]

So communication problems can affect coordination and control but should still be considered mainly as communication problems. For this reason solutions to communication problems might also positively effect coordination and control even though their main goal is to improve communication. Some problems in coordination can be solved through better communication and thus benefit from patterns that improve communication.

There are however coordination and control specific problems which they have their own solutions. Specific aspects of globally distributed software development which affect the problem areas of communication, coordination and control are temporal, geographical en socio-cultural distance. [CA01] [LLL06] As Herbsleb and Moitra state: ‘distance is a major issue in GSD (Global Software Development red.) leading to coordination, communication, and management (control, red.) problems. [HM01] Herbsleb and Moitra refer to control problems as management problems. One of the effects of distance is that it affects the frequency of communication and results in poor informal communication. [ORM04] [ORM04] The geographic distances do not even need to be global to cause problems during software development. Solutions that address communication problems of globally distributed software development projects often are useful for software development projects within the same zip-code as well. [HM01] Communication, collaboration and synchronization activities, among others, help relieving the negative influence of aspects of the globally distributed software development.

An example of a communication problem in globally distributed software development is the difficulty for employees of different geographical sites to meet face-to-face. The participants are unlikely to have unplanned contact with employees of other sites due to the absence of face-to-face, hallway or lunch conversations. In projects operating in changing environments, informal as well as formal communication is particularly important. [MH01] [CP06] As meeting face-to-face is the best way of communicating, the lack of ease to meet face-to-face has to be carefully addressed and requires special attention of the managers and developers. Various socio-cultural factors affect communication as well, for example the difference in native language between two employees makes it more difficult to communicate together, especially when they do not share the same understanding of a common language (most of the time English). Besides that other cultural differences can have their effects as well.

‘ Cultures differ on many critical dimensions, such as the need for structure, attitudes toward hierarchy, sense of time, and communication styles’. [MH01] A common syntactical language does not necessarily mean the same semantics and pragmatics. This is especially clear in the different ways of commitments and negotiations in different cultures and can lead to misunderstandings between cultures. [EN01] Employees who do not understand the culture of the employer at the other site might have difficulty in understanding them. They might feel uncomfortable contacting them or even feel insulted sometimes during communication, even though this is not necessary. [MH01] The culture does not necessarily have to be a sociological culture, it could be the company as well. Some companies support an informal way of communicating with their superiors, others require a more formal approach to communication with their superiors. If one does not understand the culture of the employers at the other site then friction could easily occur between the employees which will lead to a decreased amount of communication between the employees. This can lead to a lack of trust between sites, which can lead to a lack of willingness to communicate openly across sites.

An example of a coordination problem is the problem which occurs when a site is dependant on the artifacts produced by another site and that site has not delivered its artifact yet; or the delivered artifact does not match the expectations or specifications, quite possibly resulting in misalignment and rework. This is a result of bad coordination between sites. [HM01] The problems of coordination are very broad and can range from coordination of individual task to coordination of project goals.

An example of a control problem is the lack of knowledge of the status of a project at a specific site, therefore making it impossible to reason about the status of the entire project as well. If project managers do not get any status information on the project at a specific site they are ‘blind’ to delays and problems occurring at that site and will not be able to anticipate for any problems that will decent from that.

7. PROBLEM-GOAL-PATTERN-MEASUREMENT APPROACH

All of the patterns listed later on in this paper have a goal, as all solutions always have a goal. Each goal addresses one or multiple problems, meaning that reaching the goal benefits easing the problem addressed by the goal. Each quantitatively measurable pattern has related measurable attributes. A measurable attribute can address multiple patterns. If a
measurable attribute and its measures are clearly defined then it
can be called a metric. Using this information one can structure
the problems, goals, patterns and measurements in a graph. This
brings us up with the Problem-Goal-Pattern-Measurement
approach which will be used to structure the patterns listed later
on in this paper.

The Problem-Goal-Pattern-Measurement approach is strongly
based on the Goal-Question-Metric approach as listed in [BCR84]
[SB99] and [FP97]. The Problem-Goal-Pattern-
Measurement approach can be used to list software
development problems not specific to globally distributed
software development as well. The approach is not specific to
globally distributed software development, but is general for all
software development projects and their solution patterns.

The Problem-Goal-Pattern-Measurement approach is strongly
based on the Goal-Question-Metric approach as listed in

The Problem-Goal-Pattern-Measurement approach can be used to list software
development problems not specific to globally distributed
software development as well. The approach is not specific to
globally distributed software development, but is general for all
software development projects and their solution patterns.

8. QUANTITATIVELY MEASURABLE
PROCESS PATTERNS

Below you will find detailed descriptions of a few quantitatively
measurable process patterns. This list is by no means finite, but
is merely used as an indication of how to document
quantitatively measurable process patterns. Quantitatively
measurable process patterns are a subset of process patterns.
The specific characteristic of the patterns in this subset is that
their effect can be quantified by a metric. For the documentation
a slight modified version of the structure as used in [GHJV95]
is used. The definition of the goal of the pattern as well as the
measurable aspect of the pattern is added to the structure. The
structure resembles the structure of the Problem-Goal-Pattern-
Measurement approach. The problem descriptions describe the
problem to which the pattern is addressed, the goal descriptions
describe the goals which the patterns addresses and
the measurable aspects descriptions describe the (possible)
measurements of the pattern. This information, be it in an
abstract form, can be found in the Problem-Goal-Pattern-
Measurement graph as well. The structure also notes the related
patterns of a pattern as some patterns have strong relations. This
relation can for example be a strong resemblance, negative
influence, positive influence or an exclusion (meaning that the
implementation of the pattern excludes the implementation of
another).

By using the Problem-Goal-Pattern-Measurement (PGPM) one
can easily select the right pattern to apply. First of all an
assessment of the current situation has to be made to see which
(problem) areas require improvement the most. Then using
PGPM one can easily formulate the goals. The goals can
possibly even be quantified by setting a baseline and a
measurable target goal. [FP97, p.13] We can now see which
patterns apply to that goal. In that way the library of available
patterns can be easily searched through in order to find the
relevant patterns.

**Figure 3. Problem-Goal-Pattern-Measurement example**

By using the Problem-Goal-Pattern-Measurement (PGPM) one
can easily select the right pattern to apply. First of all an
assessment of the current situation has to be made to see which
(problem) areas require improvement the most. Then using
PGPM one can easily formulate the goals. The goals can
possibly even be quantified by setting a baseline and a
measurable target goal. [FP97, p.13] We can now see which
patterns apply to that goal. In that way the library of available
patterns can be easily searched through in order to find the
relevant patterns.

**8. QUANTITATIVELY MEASURABLE
PROCESS PATTERNS**

Below you will find detailed descriptions of a few quantitatively
measurable process patterns. This list is by no means finite, but
is merely used as an indication of how to document
quantitatively measurable process patterns. Quantitatively
measurable process patterns are a subset of process patterns.
The specific characteristic of the patterns in this subset is that
their effect can be quantified by a metric. For the documentation
a slight modified version of the structure as used in [GHJV95]
is used. The definition of the goal of the pattern as well as the
measurable aspect of the pattern is added to the structure. The
structure resembles the structure of the Problem-Goal-Pattern-
Measurement approach. The problem descriptions describe the
problem to which the pattern is addressed, the goal descriptions
describe the goals which the patterns addresses and
the measurable aspects descriptions describe the (possible)
measurements of the pattern. This information, be it in an
abstract form, can be found in the Problem-Goal-Pattern-
Measurement graph as well. The structure also notes the related
patterns of a pattern as some patterns have strong relations. This
relation can for example be a strong resemblance, negative
influence, positive influence or an exclusion (meaning that the
implementation of the pattern excludes the implementation of
another).

The diagram above describes the process of how to select and
finally implement a pattern. The fist step in the process is to
assess the current situation of the organization. From this
assessment the most critical problem areas should be identified
and goals should be chosen to solve these problem areas.
Patterns which apply to this goal should be selected from the
pattern catalogue. From this set of patterns one has to choose
the one(s) that are most suited for implementation. There are
various considerations which could affect this selection, but one
of these could be the cost of implementing a pattern. Some
patterns might be directly applicable in an organization, others
might for example need expertise which is not readily available
in the organization and thus require the acquisition of such
expertise before implementing the pattern. After having selected
the most suited pattern(s) one or more measurements for each
pattern have to be selected after which the pattern can be
implemented. The measurement(s) related to the chosen
pattern(s) might however bring one to reconsider the set goal.
The complexity of the measurement(s) might make one refrain
from using the related pattern. Some measurements might for
example require to much resources like time to measure, or
require tools which are not readily available to the organization.
One could use the measurement(s) to make the set goal
quantifiable as well. To define a quantifiable goal one has to set
a baseline and a target goal. For setting the baseline one could
use the selected measurement(s) of the pattern(s) before
applying the pattern(s) and use them again after the
implementation of the pattern(s) to check if the target goals
have been reached. This feedback loop is showed in figure 4 by
the dashed arrow. Multiple iterations of this loop can occur
before having defined the final goal, (set of) pattern(s) to be
implemented and the measurement(s) belonging to the
pattern(s).
8.1 Chunking

Pattern name: Chunking

Problem description: There is a huge need for communication and coordination between geographically distributed sites because of interdependencies between the modules developed at the different sites.

Goal(s): Reducing the interdependencies between the modules being developed; thereby reducing the interdependencies between the teams developing the modules.

Solution: Divide the software being developed in individual modules so that the amount of interdependencies between the modules is as low as possible while maintaining an acceptable cohesion within the modules. By reducing the interdependencies between modules there is less need to communicate between the developers of the modules. One should be careful though not to reduce the interdependencies between the modules without looking at the cohesion (togetherness) within the modules. Modular interdependencies and cohesion are interdependent and should be considered together when designing software. The definition and relation of cohesion and module interdependency (coupling) is explained in more detail in [DMST05]. For ease of development the modules should have an acceptable cohesion. Especially in changing environments, where modifications during development are well possible, one should keep a close eye on cohesion as modifications involving multiple developers (or modules) take much longer to complete. [MW01] If interdependencies between architectural modules exist then the developers should stay with the defined interface of their module or coordinate interface changes with the developers of dependant modules if any occur. The knowledge sharing between developers can be for the greater part limited to information about interfaces of modules. This principle is known as ‘encapsulation and information hiding’. [ORM04]

Modules without many interdependencies are well suited to be developed across geographically distributed sites, as opposed to modules with many interdependencies which should preferably be developed at the same site as much as possible or be redesigned to reduce the interdependencies.

Consequences:

Positive: A reduced number of interdependencies between modules.

An increased maintainability of the modules.

An increased reusability of the modules

Negative: Decreased cohesion within modules.

The modularity of the system under development could be hard to understand as it does not necessarily have a functional decomposition. This can decrease maintainability.

When the functional cohesion within a module is low, due to the modularization, then the reusability of the module could be low

Measurable aspect(s):

After modularization and dividing the modules over the different sites one can measure the interdependencies between the sites as follows.

\[ P = \{ S_0, ... S_j \} \]

\[ S_i = \{ M_0, ... M_q \} \]

\[ A = \{ M_q, ... M_r \} \]

\[ \bigcup P = A \]

Where \( P \) is the set of different sites, \( S_i \) is the set of modules developed at site \( S_i \), \( A \) is the set of all modules and thus represents the system under development, and the last rule ensures that all modules of the system under development are being developed at a site.

\[ f(S_i) = \sum \text{dependencies}(M_a, M_b), \]

\[ \forall (M_a \in A, M_b \in A) \]

where \( S_i \in P \) and \( (M_a \in S_i) \) and \( (M_b \notin S_i) \)

dependencies\((M_a,M_b)\) gives the number of dependencies between modules \( M_a \) and \( M_b \). The dependencies between modules can be directed, so the following rule applies.

\[ \text{dependency}(M_a, M_b) = \text{dependency}(M_b, M_a) \]

This rule means that the dependency between module \( M_a \) and \( M_b \) is not the same dependency as the dependency between module \( M_b \) and \( M_a \). Each dependency is unique.

\[ f(S_i) \] gives the total number of interdependencies between site \( S_i \) and all other sites.

If the sum is taken of the function \( f \) applied to all sites, then this gives us the total of all interdependencies between all sites; where \( A \) is the system to which the measurement is applied.

\[ \text{totalDep}(A) = \sum f(S_i), \ \forall (S_i \in P) \]

where \( S_i \in S \) and \( S_i \subseteq A \)

Result after applying the pattern:

\[ \text{totalDep}(A') < \text{totalDep}(A) \]

\( A \) is the system before the pattern is applied; and \( A' \) is the system after the pattern is applied. Thus the number of interdependencies between the sites is decreased.

This measure only quantifies the interdependencies between the modules without considering the internal cohesion of the modules. In [MM06] a metric is described which quantifies the modularization quality. The modularization quality metric considers the interdependencies between the modules as well as the internal cohesion between the modules and quantifies them according to the definition defined in [MM06].

Related pattern(s):

‘Co-location of employees’, this pattern co-locates people working on the same module. With chunking all people working on a module are at the same site as well. The ‘chunking’ pattern assign modules to co-located teams, the ‘co-location of employees’ pattern creates co-located teams to work on a module. ‘Co-location of employees’ is needed when bad modularization (or chunking) is applied.

---

1 Chunking can be done at various levels. The chunking results in modules, but these modules are variable. A module could represent a class, a package, a program module or even represent all code-units being developed at one site.
8.2 Cultural liaison

Pattern name: Cultural liaison (also known as human bridge)

Problem description: There is a lack of mutual understanding between employees of different (geographically distributed) sites. The employees are reluctant to communicate with employees at other sites and when they do there are often misunderstandings between the employees from the different sites.

Goal(s): Improve the mutual understanding of the different cultures at the different sites in order to improve the communication between the sites.

Solution: A cultural liaison is an employee who has, besides the knowledge of the culture in which he resides, adequate knowledge of another culture involved in the development process. To increase the understanding of the culture at the different sites a cultural liaison could be placed at the sites. For each different culture entailed in the development process each site should have at least one employee who deeply understands the other culture. This could be achieved by temporarily co-locating an employee at or from a different site. Sometimes a person who has knowledge of a culture because he used to live there or work for that company, site or department can be appointed as cultural liaison to avoid co-locating employees. As long as the knowledge of that person is adequate he can function as a cultural liaison. The cultural liaison has knowledge of two (or more) cultures and uses this knowledge to interconnect these cultures by increasing the mutual understanding between the both cultures. The cultural liaison assists in the communication between the cultures. In the most extreme case one could even choose to let all cross-site communication pass through a cultural liaison, but this could also greatly impede the ease of communication and cause delays in the cross-site communication as this for example renders direct cross-site communication impossible.

Consequences:

Positive: An increased mutual understanding between the sites to which the pattern is applied.

Negative: Employees might have to be (temporarily) co-located at a different site to act as a cultural liaison, which could lead to employee dissatisfaction because of the distance to his native home.

Delays in communication can occur when routing cross-site communication through the cultural liaison.

Measurable aspect(s):

To be able to measure the effects of the pattern one has to make a few assumptions on beforehand. One has to be able to ‘measure’ the culture of an employee. This can be done by defining different cultures and addressing these to the related employees. This results in a set of (pre-defined) cultures, each containing a set of employees belonging to that culture. It is possible that employees are in multiple cultures so the cultural sets are not mutually exclusive. It may be difficult, and sometimes even arguably unethical, to define distinct cultures. But, for example, in the case of organizational cultures it could be easy to define the different organizational cultures of the cooperating organizations and assign all employees to their respective organizational culture. As the differences between cultures are less obvious it becomes more difficult to value them, and this is especially the case with social cultures. The ability to value the cultures and assign these cultures to the employees determines the quality of the measurement. Even though it might be difficult to be precise and complete in the definition and assignment of cultures, it is still useful to reason about it. The measurement will allow one to estimate the size of the problem and or the effect of the applied pattern. Any measurement is better then no measurement, but one has to be aware that this measurement gives just an indication. The value of the measurement lies in the validity of the data to which it is applied. Measurements on the data can be compared as long as the categorization of cultures in the data stays the same. When the categorization is changed, then the measurements on the different sets of data can not be compared.

In an organizational graph representing the employees of a site and their relations (possibility to communicate directly) one could measure the (graph) distance from an employee to an employee within their site which has knowledge of the culture of a different site as well. The graph could have weighted edges as well which can be used to compute the weighted distance between employees. If there are no cultural liaisons then this distance is infinite. By introducing cultural liaisons this distance can be decreased. The less the distance between an employee and the cultural liaison the better and faster the communication with this cultural liaison is and thus the better the cultural understanding by that employee of the other culture can be. Overall, introducing more cultural liaisons across the organization will decrease the average distance from an employee to a cultural liaison and thus decrease the cultural distance. Of course this is not a linear relationship. One can imagine that there is an optimum amount of cultural liaisons for each project after which no added value is gained from increasing the amount of cultural liaisons; it could even harm the project to add more cultural liaison. This is however project dependant and is therefore not part of the pattern. It could also be well advised to make sure employees can reach a cultural liaison without having to communicate through higher levels in the organization then where the cultural liaison is located; this to avoid flooding the superiors with the extra workload of relaying the communication to the cultural liaison.

So this brings us up with the following rules for measurement.

\[ P = \{S_0,...,S_j\} \]
\[ E = \{E_0,...,E_i\} \]
\[ S_i \subseteq E, \forall(S_i \in P) \]
\[ C = \{C_0,...,C_v\} \]
\[ C_i \subseteq E, \forall(C_i \in C) \]

Where \( P \) is the set of different sites, \( E \) is the set of all employees, \( S_i \) is the set of employees belonging to site \( S_i \), \( C \) is the set of all cultures, and \( C_i \) is the set of employees with knowledge of the culture \( C_i \).

\[ f(C_i,C_v,S_i) = \sum_{\text{where}(E_i \in S_j) \text{ and } (E_r \in S_j) \text{ and } (E_r != E_i) \text{ and } (E_i \in C_i) \text{ and } (E_r \in C_r) \text{ and } (E_i \in C_v)} \frac{\text{dist}(E_i, E_r)}{\#S_i} \]

where \( \text{dist}(E_i, E_r) \) gives the length of the shortest path in the social network [WF94] between employee \( E_i \) and employee \( E_r \).
One employee is assigned the role of gatekeeper. Employees can be restricted. When the gatekeeper pattern is used to restrict the communications between employees it is used as some sort of formal communication channel. By letting the communication run through the gatekeeper the communication can be observed and controlled, and the direct communication between employees can be restricted.

One employee is assigned the role of gatekeeper. Employees are assigned to him/her for which he/she should regulate the communications.

**Result after applying the pattern:**

\[ f'(C_x, C_y, S_i) < f(C_x, C_y, S_i) \]

Where \( f \) is the function applied to the organization before the pattern is applied, and \( f' \) is the function applied to the organization after the pattern is applied. So the cultural distance between the cultures \( C_x \) and \( C_y \) at site \( S_i \) will decrease when applying the pattern.

**Related pattern(s):**

‘Gatekeeper’, when the cultural liaison acts as a gate for communication between two organizations then it assumes the role of gatekeeper.

8.3 Gatekeeper

**Pattern name:** Gatekeeper [Cop94, Pattern No. 23]

**Problem description:** There are isolated employees which do not have any communications with their colleagues or others; or there is too much communication between employees, leading to too much overhead in communication and decreasing the productivity of the employees. Too little as well as too much communication decreases productivity. The key is to balance communications.

**Goal(s):** Improve (balance) communications between employees of an organization.

**Solution:**

The communication between employees must be regulated. This can either mean the communication between employees must be increased and thus should be encouraged and facilitated; or that the communication must be bounded and thus should be restricted. When the gatekeeper pattern is used to solve a lack of communication between employees, then its role is to pick up introverted employees which do not have much communication. The gatekeeper then uses its communication connections and experience to assist its allocated employee(s). The gatekeeper, when assigned to introverted employees, increases their communicative capabilities.

When the gatekeeper pattern is used to restrict the communications between employees it is used as some sort of formal communication channel. By letting the communication run through the gatekeeper the communication can be observed and controlled, and the direct communication between employees can be restricted.

**Consequences:**

**Positive:** Typically introverted employees assigned to gatekeepers will be more involved and be able to deliver a greater contribution to the development process. Direct communication between employees is limited so to control the communication. This protects the employees from over-communicating which threatens their productivity. Over-communicating might lead to ‘all talk and no work’.

**Negative:** An increased workload on the employee assigned with the role of gatekeeper.

Delays in communication. Especially when the gatekeeper has difficulty managing the inflicted workload.

**Measurable aspect(s):**

The effects of the gatekeeper pattern when applied to increase communication with introverted employees can be measured as follows.

Where before application of the pattern

\[ E = \{E_0 \ldots E_a\} \]

\[ \text{relations}(E_i) = \{E_a \ldots E_d\} \]

\[ \exists(E_i) \# \text{relations}(E_i) = 0 \]

Where \( E \) is the set of employees and \( \text{relations}(E_i) \) is the set of employees with which employee \( E_i \) has a relation. A relation in this sense meaning that there is communication between the two employees. The last line implies that there is at least one employee which has no relations and is thus isolated.

After (successfully) applying the gatekeeper pattern the following rule should apply.

\[ \forall(E_i) \# \text{relations}(E_i) > 0 \]

\[ \forall(E_i) (\exists(a \text{ relations}(E_i)) \rightarrow \# \text{relations}(a) > 1) \]

The first rule meaning that there are no more isolated employees.

The second rule meaning that all employees have at least one relation with an employee which has more then one relation. This makes sure the situation where employees only communicate in pairs is excluded, so it prevents the creation of isolated pairs of employees.

**Related pattern(s):**

‘Cultural liaison’, the cultural liaison assumes the role of gatekeeper between different cultures of employees.

‘Establishment of peer-to-peer links’, the ‘Establishment of peer-to-peer links’ pattern requires the creation of gatekeepers between similar hierarchical levels of different organizations.

‘Boundary Spanner’, the boundary spanner assumes the role of gatekeeper between a customer and a software developer.
9. CONCLUSIONS
Globally distributed software development is a modern topic of research as it has not been known too long now. The literature currently devoted to globally distributed software development however supports the fact that it is an area of research which is quickly expanding, has the attention of a lot of scientists and its knowledge is still evolving. [DM06] Moreover, the many commercial companies are already using, or are interested in using, globally distributed software development as they think it brings opportunities to beat their commercial competitors. So there is a lot of commercial interest in the area of globally distributed software development as well. Global software development is becoming a norm in the software industry. [DM06] Many commercial companies started using globally distributed software development not too long ago because of its potential to reduce development costs, reduce the time-to-market, increase their local knowledge and increase their employment pool among many other reasons. It has become clear that these advantages can be achieved but that they do not come easy. Globally distributed software development brings many specific problems with it which have to be addressed in order to achieve the advantages it can bring. As the problems occurring during globally distributed software development are often recurring across various different projects there have been documented many solutions to these problems. If abstracted to a general solution these solutions can be documented as a pattern to support reuse of the solution in similar situations. By documenting ‘best practices’ as patterns the knowledge body of globally distributed software development increases and makes the process of development better understood and thus less treacherous to apply in a company. The documenting of knowledge in patterns also makes the knowledge more accessible and applicable for others, for example companies facing the same problems.

Due to the many possible and documented solutions it is difficult to find the right solution for a given problem. By documenting the solutions described in the literature as patterns it becomes more easy to reuse the existing or gained knowledge. This makes the solutions more easily applicable, but still makes it difficult to find the relevant solutions for a given problem as the library of solution patterns is vast. Therefore I suggested to structure the patterns using the Problem-Goal-Pattern-Measurement approach. By using this approach it is more easy to keep an overview of the available knowledge and more easy to find the applicable solution patterns for a given problem. By defining the goals of the patterns it is also more clear what the patterns try to achieve.

Quantitatively measurable patterns are the patterns of which the effects can be measured in a quantitative way. These measures can be used to assess the current situation, predict the future when applying a pattern and measure the true effects of applying a pattern to see whether its application is successful or not. Measuring clearly helps project management to manage a project as they can assess the current situation, set goals for the future and act accordingly. These quantitatively measurable patterns can also be implemented in tools which can assess data and come up with possible process patterns which could improve the situation. Such a tool would greatly reduce the effort needed to find the right solution patterns and make these patterns therefore more easily applicable.

So it is made clear that it is very useful to be able to quantitatively measure aspects which are influenced by a process pattern to quantify the effects of the solution as well as be able to assess the need to apply the solution. By using the Problem-Goal-Pattern-Measurement the applicable patterns and their measurements are easily found for each problem and their goals are made clear.

Globally distributed software development has great potential for sure, but one should never forget that each opportunity also brings its risk. These risks should be addressed with caution; and quantitatively measurable process patterns assist in clearing the pitfalls on the road of globally distributed software development. One should always keep in mind though that no amount of management technique or process organization can ever overcome a lack of ability or talent by employees, ranging from architects (software designers) up until software programmers.

10. FUTURE WORK
For now the Problem-Goal-Pattern-Measurement approach is only a way to structure the knowledge body of patterns. This approach could however be more formalized by formalizing goal templates for example. This could help structure the Problem-Goal-Pattern-Measurement approach and therefore give more structure to the knowledge body of solution patterns as a whole.

The list of a few solution patterns described in this paper is far from complete. It is by no means meant to be complete as well, as the knowledge body of solution patterns is constantly growing. Every problem has numerous solutions, so numerous patterns can be documented to solve a problem; each in its own way. It would be of great significance to expand the current Problem-Goal-Pattern-Measurement framework with more documented solution patterns. These patterns do not necessarily have to be specific to globally distributed software development problems but can address any software development problems existing.

Implementing the quantitatively measurable process patterns in a process analyzing tool would make the process patterns even more accessible and easily applicable. The quantitative metrics can be used to analyze the data and point out areas where possible problems could occur. If these potential problem areas can be identified soon and the relevant solution patterns can be found and assessed easily, then bigger problems can be avoided by timely acting. This would greatly reduce the risks of globally distributed software development and thus make it more interesting to use.

REFERENCES


