Intertwingularity Mapping

wiki

https://en.wikipedia.org/wiki/Wiki

A wiki is run using wiki software, otherwise known as a wiki engine. A wiki engine is a type of content management system, but it differs from most other such systems, including blog software, in that the content is created without any defined owner or leader, and wikis have little inherent structure, allowing structure to emerge according to the needs of the users.

A wiki is essentially a database for creating, browsing, and searching through information. A wiki allows non-linear, evolving, complex, and networked text

Within the text of most pages, there are usually many hypertext links to other pages within the wiki. This form of non-linear navigation is more "native" to a wiki than structured/formalized navigation schemes. Users can also create any number of index or table-of-contents pages, with hierarchical categorization or whatever form of organization they like.

Wikis can provide one or more ways to categorize or tag pages to support the maintenance of such index pages. Some wikis, including the original, have a backlink feature, which displays all pages that link to a given page. It is also typically possible in a wiki to create links to pages that do not yet exist, as a way to invite others to share what they know about a subject new to the wiki. Wiki users can typically "tag" pages with categories or keywords, to make it easier for other users to find the article. For example, a user creating a new article on cold weather cycling might "tag" this page under the categories of commuting, winter sports and bicycling. This would make it easier for other users to find the article.

Modular Design

https://en.wikipedia.org/wiki/Modular_design

Modular design, or "modularity in design", is an approach (design theory and practice) that subdivides a system into smaller parts called modules or skids, which can be independently created, modified, replaced or exchanged between different systems. A modular design can be characterized by functional partitioning into discrete scalable, reusable modules; rigorous use of well-defined modular interfaces; and making use of industry standards for interfaces.

Transclusion

https://en.wikipedia.org/wiki/Transclusion

In computer science, transclusion is the inclusion of part or all of an electronic document into one or more other documents by hypertext reference. Transclusion is usually performed when the referencing document is displayed, and is normally automatic and transparent to the end user.[1] The result of transclusion is a single integrated document made of parts assembled dynamically from separate sources, possibly stored on different computers in disparate places.

Transclusion facilitates modular design: a resource is stored once and distributed for reuse in multiple documents. Updates or corrections to a resource are then reflected in any referencing documents. Ted Nelson coined the term for his 1980 nonlinear book Literary Machines, but the idea of master copy and occurrences was applied 17 years before, in Sketchpad.

Hypertext

https://en.wikipedia.org/wiki/Hypertext

Hypertext documents can either be static (prepared and stored in advance) or dynamic (continually changing in response to user input, such as dynamic web pages). Static hypertext can be used to cross-reference collections of data in documents, software applications, or books on CDs. A well-constructed system can also incorporate other user-interface conventions, such as menus and command lines. Links used in a hypertext document usually replace the current piece of hypertext with the destination document. A lesser known feature is StretchText, which expands or contracts the content in place, thereby giving more control to the reader in determining the level of detail of the displayed document. Some implementations support transclusion, where text or other content is included by reference and automatically rendered in place.

Hypertext can be used to support very complex and dynamic systems of linking and cross-referencing. The most famous implementation of hypertext is the World Wide Web, written in the final months of 1990 and released on the Internet in 1991.

HyperText is a way to link and access information of various kinds as a web of nodes in which the user can browse at will.

Chunking

https://en.wikipedia.org/wiki/Chunking_(psychology)

In cognitive psychology, chunking is a process by which individual pieces of an information set are broken down and then grouped together.[1] A chunk is a collection of basic familiar units that have been grouped together and stored in a person's memory. These chunks are able to be retrieved more easily due to their coherent familiarity.[2] It is believed that individuals create higher order cognitive representations of the items within the chunk. The items are more easily remembered as a group than as the individual items themselves. These chunks can be highly subjective because they rely on an individuals perceptions and past experiences, that are able to be linked to the information set. The size of the chunks generally range anywhere from two to six items, but often differ based on language and culture.

A modality effect is present in chunking. That is, the mechanism used to convey the list of items to the individual affects how much "chunking" occurs.

Experimentally, it has been found that auditory presentation results in a larger amount of grouping in the responses of individuals than visual presentation does. Previous literature, such as George Miller's The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information (1956) has shown that the probability of recall of information is greater when the "chunking" strategy is used. As stated above, the grouping of the responses occurs as individuals place them into categories according to their inter-relatedness based on semantic and perceptual properties. Lindley (1966) showed that since the groups produced have meaning to the participant, this strategy makes it easier for an individual to recall and maintain information in memory during studies and testing.[4] Therefore, when "chunking" is used as a strategy, one can expect a higher proportion of correct recalls.

https://en.wikipedia.org/wiki/The_Magical_Number_Seven,_Plus_or_Minus_Two

Conceptual graph

https://en.wikipedia.org/wiki/Conceptual_graph

Morphological Analysis

https://en.wikipedia.org/wiki/Morphological_analysis_(problem-solving)

As a problem-structuring and problem-solving technique, MA was designed for multi-dimensional, non-quantifiable problems where causal modelling and simulation do not function well, or at all.

Information Mapping

https://en.wikipedia.org/wiki/Information_mapping

analyze, organize and present deeply intertwingled information

Chunking
    Break up information into small, manageable units
Relevance
    Limit each unit of information to a single topic
Labeling
    Label each unit of information in a way that identifies its contents
Consistency
    Be consistent in use of terminology as well as in organizing, formatting and sequencing information
Accessible detail
    Organize and structure information so those who need detail can access it easily, while those who don't can easily skip it

Quick, easy access to information at the right level of detail, even for diverse audiences

Dashboards

https://en.wikipedia.org/wiki/Dashboard_(business)

A dashboard is a type of graphical user interface which often provides at-a-glance views of key performance indicators (KPIs) relevant to a particular objective or business process. In other usage, "dashboard" is another name for "progress report" or "report."

The "dashboard" is often displayed on a web page which is linked to a database that allows the report to be constantly updated. For example, a manufacturing dashboard may show numbers related to productivity such as number of parts manufactured, or number of failed quality inspections per hour. Similarly, a human resources dashboard may show numbers related to staff recruitment, retention and composition, for example number of open positions, or average days or cost per recruitment.[1]

Evolutionary Database Design

https://en.wikipedia.org/wiki/Evolutionary_database_design

Database refactoring
Evolutionary data modeling

Database Normalization

https://en.wikipedia.org/wiki/Database_normalization

Database normalization is the process of structuring a relational database [clarification needed] in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity

Normalization entails organizing the columns (attributes) and tables (relations) of a database to ensure that their dependencies are properly enforced by database integrity constraints. It is accomplished by applying some formal rules either by a process of synthesis (creating a new database design) or decomposition (improving an existing database design).

Agile Documentation

http://agilemodeling.com/essays/agileDocumentation.htm

The fundamental issue is communication, not documentation.

Take an evolutionary approach to documentation development, seeking and then acting on feedback on a regular basis.

Well-written documentation supports organizational memory

Documentation should be concise: overviews/roadmaps are generally preferred over detailed documentation.

Travel as light as you possibly can.

The benefit of having documentation must be greater than the cost of creating and maintaining it.

Create documentation only when you need it

Update documentation only when it hurts.

When do we slice'n dice? To support communication, to support organizational memory, to understand something.

Agile documents are "lean and sufficient".
Agile documents fulfill a purpose.
Agile documents describe "good things to know".
Agile documents are sufficiently accurate, consistent, and detailed.

Keep documentation just simple enough, but not too simple
Write the fewest documents with least overlap
Put the information in the most appropriate place
Display information publicly
Document with a purpose
Focus on the needs of the actual customers(s) of the document
Iterate, iterate, iterate (evolutionary approach: incremental)
Start with models you actually keep current
Update only when it hurts more not to

Agile Modeling

http://agilemodeling.com/

Single Source Information

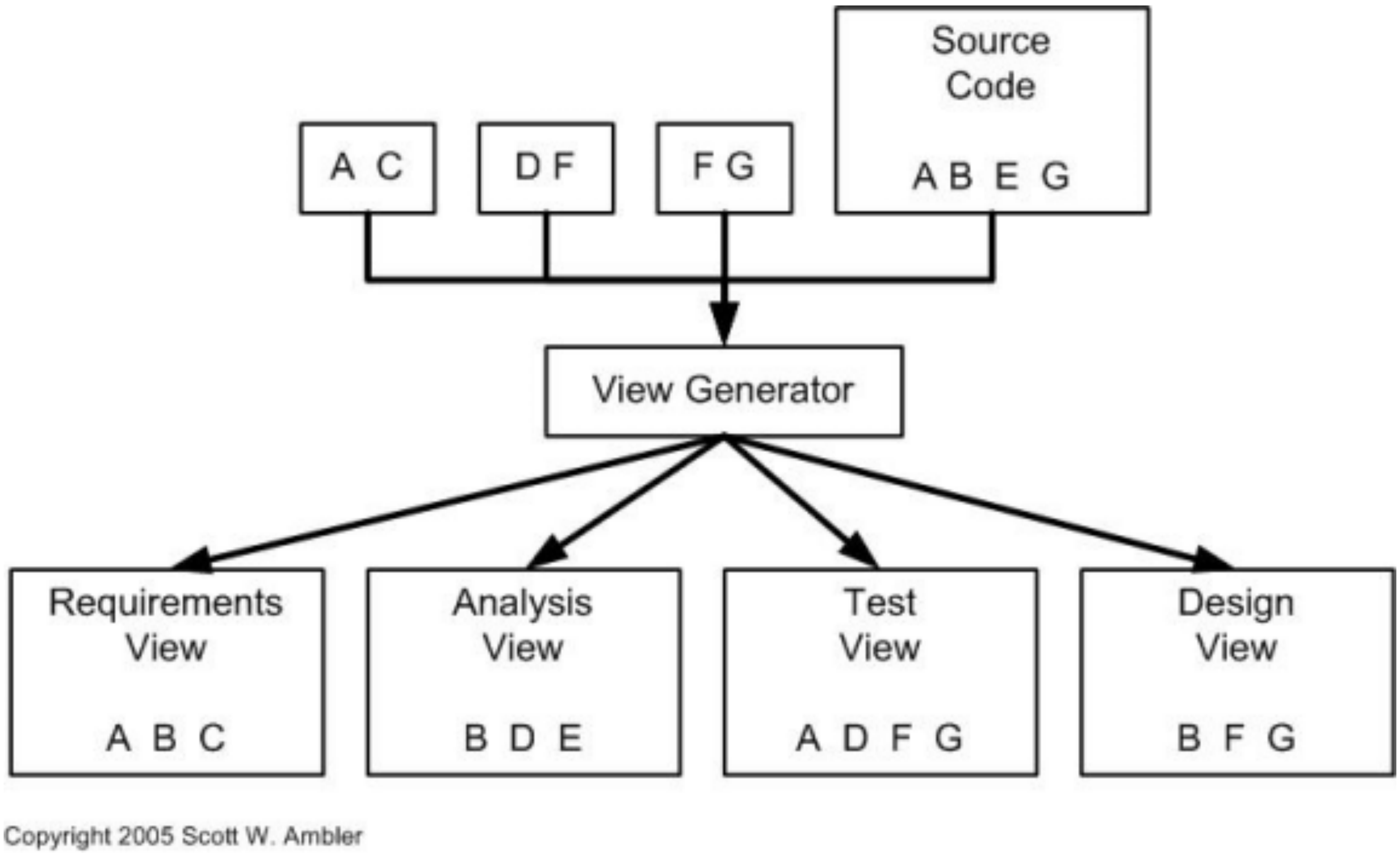
Why do you want to record a concept once? Three reasons:

Reduce your maintenance burden. The more representations that you maintain the greater the maintenance burden because you'll want to keep each version in sync with each other. Figure 1 depicts a typical approach to traditional software development artifacts. The letters in each artifact represent a piece of information stored within it. For example, information A (perhaps our business rule mentioned above) is captured in the requirements document, test model, and source code.
Reduce your traceability burden. With multiple copies the greater your traceability needs will be because you'll need to relate each version to its alternate representations, otherwise you'll never be able to keep them synchronized when a change does occur. Yes, AM advises you to update only when it hurts but the more copies you have of something the more likely it is that it will start to hurt earlier.
Increase consistency. The more copies you have the greater the chance you will have inconsistent information because you very likely won't be able to keep the versions synchronized.

Views, Not Copies
It is clear that you should store system information in one place and one place only, ideally in the most effective place. From the point of view of software development this concept is called normalization, in the programming world an important aspect of literate programming, and in the technical documentation world it is called single sourcing. With single sourcing the idea is to record technical information only once and then generate various documentation views as needed from that information. In the case of the business rule example above it would be recorded using some sort of business rule definition language. A human readable view would be generated for your requirements documentation (this is easier said than done, by the way, but for the sake of argument let's assume that it's possible) and an implementation view generated which would

either be run by your business rule engine or compiled as application source code.

Figure 2 depicts a strategy for single sourcing all of the information contained in Figure 1, then automatically generating the original artifacts of Figure 1 through the use of a generator. The views are generated on an as needed basis from the most recent source information, ensuring that they are up-date as of the generation time. .



An important implication of Figure 2 is that although the information is stored in a single place, it can be rendered in multiple ways for different audiences. This is called the Locality of Reference Documentation Principle. End users will need to see information in a different format than programmers, for example. Some people prefer to see diagrams whereas others prefer information in textual form. Just because information needs to be viewed, and worked with, in multiple ways doesn't mean that it needs to be stored multiple times. Just as you build working software from your source code base, you would "build" your documentation views from your single-sourced information base.

Document (Structure?) Late

http://agilemodeling.com/essays/documentLate.htm

Well, not late, but rather structure as needed when needed. So don't worry about structure until structures are needed.

Document Continuously

http://agilemodeling.com/essays/documentContinuously.htm

Multiple Models

http://agilemodeling.com/artifacts/

To be effective, the principle Multiple Models tells us that agile modelers should know a wide variety of modeling techniques so that they have the skills and knowledge to apply the right artifact(s) for the situation at hand. Unfortunately this is easier said than done.

Just Barely Good Enough

http://agilemodeling.com/essays/barelyGoodEnough.html

Complex System

Complex systems are systems whose behavior is intrinsically difficult to model due to the dependencies, competitions, relationships, or other types of interactions between their parts or between a given system and its environment. Systems that are "complex" have distinct properties that arise from these relationships, such as nonlinearity, emergence, spontaneous order, adaptation, and feedback loops, among others.

The term complex systems often refers to the study of complex systems, which is an approach to science that investigates how relationships between a system's parts give rise to its collective behaviors and how the system interacts and forms relationships with its environment.[1] The study of complex systems regards collective, or system-wide, behaviors as the fundamental object of study; for this reason, complex systems can be understood as an alternative paradigm to reductionism, which attempts to explain systems in terms of their constituent parts and the individual interactions between them.

As an interdisciplinary domain, complex systems draws contributions from many different fields, such as the study of self-organization from physics, that of spontaneous order from the social sciences, chaos from mathematics, adaptation from biology, and many others.

Complex systems are chiefly concerned with the behaviors and properties of systems. A system, broadly defined, is a set of entities that, through their interactions, relationships, or dependencies, form a unified whole. It is always defined in terms of its boundary, which determines the entities that are or are not part of the system. Entities lying outside the system then become part of the system's environment.

A system can exhibit properties that produce behaviors which are distinct from the properties and behaviors of its parts; these system-wide or global properties and behaviors are characteristics of how the system interacts with or appears to its environment, or of how its parts behave (say, in response to external stimuli) by virtue of being within the system.

Another common feature of complex systems is the presence of emergent behaviors and properties: these are traits of a system that are not apparent from its components in isolation but which result from the interactions, dependencies, or relationships they form when placed together in a system.

When emergence describes the appearance of unplanned order, it is spontaneous order (in the social sciences) or self-organization (in physical sciences). Spontaneous order can be seen in herd behavior, whereby a group of individuals coordinates their actions without centralized planning. Self-organization can be seen in the global symmetry of certain crystals, for instance the apparent radial symmetry of snowflakes, which arises from purely local attractive and repulsive forces both between water molecules and between water molecules and their surrounding environment.

Complexity in practice

The traditional approach to dealing with complexity is to reduce or constrain it. Typically, this involves compartmentalization: dividing a large system into separate parts. Organizations, for instance, divide their work into departments that each deal with separate issues. Engineering systems are often designed using modular components. However, modular designs become susceptible to failure when issues arise that bridge the divisions.

Channel Capacity

Channel capacity, in electrical engineering, computer science, and information theory, is the tight upper bound on the rate at which information can be reliably transmitted over a communication channel.

Following the terms of the noisy-channel coding theorem, the channel capacity of a given channel is the highest information rate (in units of information per unit time) that can be achieved with arbitrarily small error probability.

Three degrees of influence

https://en.wikipedia.org/wiki/Three_degrees_of_influence

Christakis and Fowler explored the influence of social connections on behavior. They described how social influence does not end with the people to whom a person is directly connected. People influence their friends, who in turn influence their friends, and so on; hence, a person's beliefs and actions can influence people he or she has never met, to whom he or she is only indirectly tied. Christakis and Fowler posited that diverse phenomena "ripple through our network, having an impact on our friends (one degree), our friends’ friends (two degrees), and even our friends’ friends’ friends (three degrees). Our influence gradually dissipates and ceases to have a noticeable effect on people beyond the social frontier that lies at three degrees of separation." They posited a number of reasons for this decay, and they offered informational, psychological, and biological rationales.

This argument is basically that peer effects need not stop at one degree of separation. However, across a broad set of empirical settings, using both observational and experimental methods, they observed that the effect seems, in many cases, to no longer be meaningful at a social horizon of three degrees.

Influence dissipates after three degrees (to and from friends’ friends’ friends) for three reasons, Christakis and Fowler propose:[2]

Intrinsic decay -- corruption of information, or a kind of "social friction" (like the game telephone).
Network instability -- social ties become unstable (and are not constant across time) at a horizon of more than three degrees of separation.
Evolutionary purpose -- we evolved in small groups where everyone was connected by three degrees or fewer (an idea receiving subsequent support

Six degrees of separation

Six degrees of separation is the idea that all people are six, or fewer, social connections away from each other.

the modern world was 'shrinking' due to this ever-increasing connectedness of human beings. He posited that despite great physical distances between the globe's individuals, the growing density of human networks made the actual social distance far smaller.

degree of connectedness

Information Theory

the transmission, processing, extraction, and utilization of information

Information Sciences

Accessibility and Usability of Information

origination, collection, organization, storage, retrieval, interpretation, transmission, transformation, and utilization of information

collection, classification, manipulation, storage, retrieval and dissemination of information

Information Systems

Knowledge management

Knowledge Acquisition and

Documentation Structuring

https://commonkads.org/

Knowledge has come to be recognized and handled as a valuable entity in itself. Surveys consistently show that top executives consider knowledge to be the single most important factor in organizational success. In this knowledge-driven society, knowledge systems have their place as an important mainstream technology. That is why there is a strong need to convert the art and craft of knowledge engineering into a real scientific discipline.

Model-driven architecture

Guidelines for structuring

Agile Modeling

Document Continuously

Single-source information

Just Barely Good Enough

Agile Software Development

adaptive, iterative and evolutionary development

Refactoring

dynamic, non-deterministic and non-linear characteristics

Emergent requirements versus big requirements up front

Traceability

Unified Software Development

Iterative and incremental

Adaptive Software Development

Continuous Adaptation

to emergent ...

Iterative

Change Tolerant

Scrum

Incremental

Lean Software Development

Iterative

Refactoring

Decide as late as possible

Adapt to Changes

Correct Mistakes

Deliver as fast as possible

Rapid Application Development

Adaptive Process

Agile Process

RAD Approaches

Adaptive Model

Agile Model

Unified Model

Component Content Management System

https://en.wikipedia.org/wiki/Component_content_management_system#Sources

A component content management system (CCMS) is a content management system that manages content at a granular level (component) rather than at the document level. Each component represents a single topic, concept or asset (for example an image, table, product description, a procedure).

Each component is only stored one time in the content management system, providing a single, trusted source of content (referential). These components are then reused (rather than copied and pasted) within a document or across multiple documents. This ensures that content is consistent across the entire documentation set.[2] The use of components can also reduce the amount of time it takes to update and maintain content as changes only need to be made once, in one component.

Components can be as large as a chapter or as small as a definition or even a word. Components in multiple content assemblies (content types) can be viewed as components or as traditional documents.

Modular Documentation

https://redhat-documentation.github.io/modular-docs/#:~:targetText=Modular%20documentation%20is%20documentation%20based,can%20also%20include%20other%20assemblies.&targetText=For%20definitions%20of%20the%20terms,Modular%20Documentation%20Terms%20and%20Definitions.

Modular documentation is documentation based on modules, which the writer combines into assemblies. An assembly can also include other assemblies.

Componentization for Knowledge

https://blog.okfn.org/2007/04/30/what-do-we-mean-by-componentization-for-knowledge/#:~:targetText=Componentization%20is%20the%20process%20of,%2C%20at%20present%2C%20least%20advanced.

Componentization is the process of atomizing (breaking down) resources into separate reusable packages that can be easily recombined.

Componentization is the most important feature of (open) knowledge development as well as the one which is, at present, least advanced. If you look at the way software has evolved it now highly componentized into packages/libraries. Doing this allows one to ‘divide and conquer’ the organizational and conceptual problems of highly complex systems. Even more importantly it allows for greatly increased levels of reuse.

The power and significance of componentization really comes home to one when using a package manager (e.g. apt-get for debian) on a modern operating system. A request to install a single given package can result in the automatic discovery and installation of all packages on which that one depends. The result may be a list of tens — or even hundreds — of packages in a graphic demonstration of the way in which computer programs have been broken down into interdependent components.

The Four Principles of (Open) Knowledge Development

https://blog.okfn.org/2006/05/09/the-four-principles-of-open-knowledge-development/

Open knowledge allows (and requires for its success) a development process that is:

Incremental
Decentralized (and asyncrhonous)
Collaborative
Componentized (and ‘packagized’)

Incremental implies a process that allows for lots of gradual improvement and contribution. Like all of the the four principles this applies to the development of closed as well as open knowledge. However it operates more powerfully in an ‘open’ situation where the boundary between participants and non-participants is porous and it is easier for ‘just anyone’ to get involved.

Open knowledge allows for a decentralized (and asynchronous) development process. This greatly reduces rigidities and also allows for a far wider participation in a given project. At the same time it demands a more sophisticated set of development tools and processes.

Just like code, knowledge can be developed by a single individual but its real strength is that it can be developed collaboratively. The collaborative aspect of open knowledge is strongly related, and dependent upon, the the principles of decentralization and componentization.

This probably the most important feature of (open) knowledge development as well as the one which is, at present, least advanced. If you look at the way software has evolved it now highly componentized into packages/libraries. Doing this allows one to ‘divide and conquer’ the organizational and conceptual problems of highly complex systems. Even more importantly it allows for greatly increased levels of reuse.

The power and significance of componentization really comes home to one when using a package manager (e.g. apt-get for debian) on a modern operating system. A request to install a single given package can result in the automatic discovery and installation of all packages on which that one depends. The result may be a list of tens — or even hundreds — of packages in a graphic demonstration of the way in which computer programs have been broken down into interdependent components.