ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THÈSE PAR ARTICLES PRÉSENTÉE À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DU
DOCTORAT EN GÉNIE
Ph.D.

PAR
Sébastien RUFIANGE

VISUALISATIONS NOVATRICES POUR LA COMPRÉHENSION DE RÉSEAUX ET DE LOGICIELS COMPLEXES

MONTRÉAL, LE 26 AOÛT 2013

PRÉSENTATION DU JURY

CETTE THÈSE A ÉTÉ ÉVALUÉE

PAR UN JURY COMPOSÉ DE:

M. Christopher P. Fuhrman, directeur de thèse
Département de génie logiciel et des TI, ÉTS

M. Michael J. McGuffin, codirecteur
Département de génie logiciel et des TI, ÉTS

M. Adel Francis, président du jury
Département de génie de la construction, ÉTS

Mme. Anastasia Bezerianos, examinateur externe
Laboratoire de Recherche en Informatique, Université Paris-Sud

M. David Labbé, membre du jury
Département de génie logiciel et des TI, ÉTS

ELLE A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 28 JUIN 2013

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens d'abord à remercier mes directeurs qui m'ont accompagné et soutenu tout au long de cette thèse. Cris, j'ai pu apprécier tes qualités de chercheur et de superviseur. Tu étais toujours disponible et motivé à m'écouter, m'encourager et je t'en suis sincèrement reconnaissant. J'ai aimé nos conversations, parfois passionnées, parfois un peu moins sérieuses, ainsi que ta grande générosité. Merci pour ton support constant.

Michael, grâce à toi, j'ai enrichi mes expériences en recherche, assisté à des conférences et visité des entreprises (Toronto, IHM, VisWeek). J'ai bénéficié, personnellement et professionnellement, de ton expérience, de tes contacts et de ta passion pour la recherche. J'ai beaucoup apprécié nos discussions, nos voyages et nos journées de rédaction parfois plus remplies. Je savais que je pouvais toujours compter sur toi. Ton écoute et ton enthousiasme ont été pour moi une source de motivation. J'ai aimé les moments plus amusants que nous avons passé ensemble. Merci sincèrement à vous deux, je me trouve chanceux de vous avoir eu comme directeurs.

Cette thèse n'aurait pas pu être écrite aussi facilement si je n'avais pas été supporté financièrement par des organismes, ainsi que par mes directeurs de recherche. Je tiens à remercier le Fonds de recherche du Québec, l'ÉTS et les entreprises qui m'ont appuyé.

J'aimerais très sincèrement remercier les membres du jury, Adel Francis, Anastasia Bezerianos, David Labbé, d'avoir accepté d'évaluer ma thèse. Je tiens aussi à remercier le groupe de recherche HIFIV pour nos échanges et leur participation à mes expériences.

Durant mes études, j'ai parfois dû négliger mes proches. Je remercie mes amis pour leurs encouragements soutenus et leur compréhension. Un grand merci à ma famille et ma belle-famille (Josianne, Pierre, Daphnée, Marie et Antoine) pour leur support inconditionnel. J'aimerais aussi plus particulièrement remercier mon amoureuse, Marie-Ève. Ta tendresse et ta patience m'ont aidé à réaliser cette thèse avec plus de facilité. Merci à tous !

VISUALISATIONS NOVATRICES POUR LA COMPRÉHENSION DE RÉSEAUX ET DE LOGICIELS COMPLEXES

Sébastien RUFIANGE

RÉSUMÉ

La visualisation d'information a le potentiel de pouvoir exploiter nos capacités visuelles, acquises au fil de centaines de millions d'années d'évolution, afin de faciliter la découverte de secrets enfouis dans les données, de nouveaux patrons ou de relations insoupçonnées. Il existe toutefois une grande variété de données, plus ou moins structurées, que l'on cherche à comprendre sous diverses perspectives.

En particulier, les données sous forme de réseaux servent à modéliser des phénomènes importants, tels que les communautés sociales ou les transactions financières, mais peuvent être difficiles à représenter si les réseaux sont grands, hiérarchiques, et/ou dynamiques. Cette thèse se concentre sur la conception de nouvelles techniques de visualisation de réseaux, dans le but de faciliter la compréhension de données.

Les techniques de visualisation présentes dans la littérature sont utiles dans certains contextes et comportent chacune des limitations. Néanmoins, il existe encore des possibilités inexplorées pour créer des nouvelles façons de représenter des données. La validation de ces nouvelles techniques demeure un défi. En outre, les interfaces doivent être simples à utiliser, mais aussi faciliter l'analyse et l'exploration de données.

Dans le but d'étudier de nouvelles options de visualisations pour faciliter des tâches de compréhension des données, nous avons d'abord classifié les travaux antérieurs avec des taxonomies. De cette manière, nous avons aussi pu mettre en lumière des nouvelles pistes d'hybrides (c'est-à-dire, des combinaisons d'approches) potentiellement intéressantes pour visualiser des réseaux statiques et dynamiques.

Les contributions présentées dans cette thèse couvrent différents aspects de la visualisation de réseaux complexes et dynamiques. D'abord, le premier chapitre se concentre sur la visualisation de réseaux statiques comportant des hiérarchies, par la combinaison d'approches. Le prototype décrit dans le deuxième chapitre permet également de combiner des représentations visuelles, mais peut être aussi utilisé afin de modéliser des graphes dynamiques. Enfin, le troisième chapitre présente une nouvelle méthode visuelle appliquée afin de tracer l'évolution de structures de conception complexes dans des logiciels (modélisés par des réseaux).

Ainsi, dans le premier prototype (TreeMatrix), des parties de graphes sont montrées avec des matrices et des diagrammes noeuds-liens, alors que les arborescences sont représentées par des diagrammes en glaçons et des regroupements. Contrairement aux autres visualisations dans la littérature, cette nouvelle technique aide à montrer des réseaux denses, sans nuire à la compréhension des liens à plus haut niveau. Une expérience avec des utilisateurs a montré certains

avantages afin de découvrir et organiser les liens de modules au sein d'un logiciel, en comparaison avec le logiciel commercial Lattix.

Nous avons également combiné des approches de manière novatrice pour notre second prototype (DiffAni) afin de visualiser des réseaux qui évoluent dans le temps. DiffAni est le premier hybride interactif de graphes dynamiques et sa validation avec des participants a permis de faire ressortir certains avantages. Ainsi, l'utilisation d'animation doit être modérée et est surtout utile lors de mouvements significatifs. Ces résultats, avec nos taxonomies, pourraient contribuer à guider la création de nouveaux hybrides dans le futur.

Le troisième prototype (IHVis) a facilité l'exploration et le traçage de structures de conception dans des logiciels en évolution (modélisés par des réseaux) à partir de répertoires de code source. Cette nouvelle visualisation a notamment révélé des cas d'introduction de points de stabilité et des *refactorings*, et certains participants ont aussi trouvé d'autres informations intéressantes, telles que l'extension de fonctionnalités par l'implémentation d'interfaces.

En résumé, cette thèse présente des façons novatrices et utiles de visualiser des réseaux complexes et dynamiques. Nos principales contributions sont (1) l'exploration d'espaces de conception de nouvelles visualisations de réseaux à l'aide de taxonomies, (2) la conception de prototypes combinant des approches pour visualiser des réseaux hiérarchiques et dynamiques, (3) la conception d'une nouvelle méthode visuelle d'exploration des variations et des instabilités au sein de logiciels en évolution, (4) l'évaluation de ces techniques à l'aide d'expériences avec des participants.

Mot-clés : visualisation d'information, réseau, taxonomie, visualisation hybride, visualisation de logiciels

NOVEL VISUALIZATIONS TO UNDERSTAND COMPLEX NETWORKS AND SOFTWARE

Sébastien RUFIANGE

ABSTRACT

The presentation of information can potentially benefit from our biological visual system, which evolved over hundreds of millions of centuries, to facilitate the discovery of hidden secrets, new patterns or unexpected relationships in data. There is much information in the world, of various kinds, structured or unstructured, that we seek to understand from different points of view. In the literature, visualization techniques can be useful in specific contexts, but each has its own limitations. However, there are still many opportunities to create and explore new ways of representing data and networks. The validation of these novel techniques remains a challenge. Indeed, these new interfaces must be easy to use, but at the same time, help users analyze and explore complex data.

In particular, important data can be modeled as networks, such as social interactions or financial transactions, but it is harder to visualize these networks if they are large, include hierarchies and/or evolve over time. This thesis focuses on designing new visualization techniques of networks to facilitate the understanding of data.

In the goal of comparing alternative visualization techniques to support data comprehension tasks, we have classified previous work using taxonomies. We were thus able to reveal new interesting combinations (or hybrids) to visualize static and dynamic networks.

Our contributions cover different aspects of the visualization of complex and dynamic graphs. The first chapter focuses on visualizing static compound graphs by combining existing techniques. The prototype that we describe in the second chapter also explores combinations of representations, but can be used to show dynamic graphs. Finally, the third chapter present a new visual and practical approach to track complex software structures in evolving software designs (modeled as networks).

In our first prototype (TreeMatrix), parts of a network are shown using matrices and node-link diagrams, while tree structures are shown using icicle diagrams and nested enclosures. One benefit of this technique is that it helps visualize dense networks, without affecting the ability to comprehend higher level relationships. A user study has shown advantages of using our novel technique to discover and organize links in a software design, compared to a mature commercial tool (Lattix).

We also combined techniques in novel ways in our second prototype (DiffAni) to visualize evolving networks. DiffAni is the first hybrid approach that can be used for dynamic networks, and its validation using human subjects revealed benefits in some cases. Thus, animation should

be used sparingly, i.e., when there is significant movement in a network. These results along with our taxonomies could guide the exploration of more novel combinations in future work.

Finally, our third prototype (IHVis) focused on exploring and tracking evolving software design structures. Using data collected in software source code repositories, our new visual tool was able to discover interesting cases of stability protection, and *refactorings*. Also, several human subjects reported their findings, such as the implementation of new variations in features.

In summary, this thesis present novel and useful ways to show complex and evolving network structures. Our main contributions are (1) the exploration of design spaces for novel network visualizations using taxonomies, (2) implementation of prototypes combining techniques to visualize compound graphs and dynamic networks, (3) a new visual method to explore extensions and instabilities in evolving software, (4) the evaluation of these techniques in user studies.

Keywords: information visualization, network, taxonomy, hybrid visualization, software visualization

TABLE DES MATIÈRES

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CVS	Concurrent Versions System
SVN	SubVersioN
UML	Unified Modeling Language
OO	Object Oriented
OOD	Object Oriented Design
LOC	Lines Of Code
SM	Small Multiples

INTRODUCTION

La quantité d'information numérique dans le monde atteignait 1,8 zétaoctets (10^{21} octets ou 1800 milliards de gigaoctets) en 2011 et croit plus vite que la Loi de Moore (double tous les deux ans) [1]. Ces données sont de plus en plus complexes et diversifiées, ce qui rend aussi leur analyse plus difficile. Néanmoins, de nombreuses décisions sont prises chaque jour en fonction de l'interprétation de ces informations. Afin d'aider à mieux gérer cette complexité, la visualisation d'information (ou *infovis*) peut faciliter la compréhension des données en exploitant les capacités d'analyses visuelles naturelles des humains (Ware, 2000; Mazza, 2009). En outre, les données structurées sous forme de réseaux sont omniprésentes dans notre société et peuvent modéliser des interactions sociales, des échanges commerciaux, des couplages entre des modules de code logiciels, des réseaux de communication, des systèmes biologiques et des migrations de populations.

Il existe diverses techniques de visualisation dans la littérature et elles ont chacune des contraintes et contextes d'utilisation spécifiques. Malheureusement, le type de données, la réponse recherchée, la complexité, le domaine d'application, ainsi que d'autres facteurs peuvent compliquer le choix de la visualisation appropriée (Mackinlay *et al.*, 2007). Or, l'utilisation d'une visualisation moins adéquate peut contribuer à réduire notre capacité à déceler et interpréter des observations intéressantes dans certaines situations. Aussi, il manque actuellement des évaluations d'approches, dans le but d'aider à choisir une représentation plutôt qu'une autre, dans un contexte donné. Les chercheurs doivent souvent opter pour une technique ou une autre, alors qu'il pourrait être avantageux, dans certains cas, de concevoir de nouvelles techniques adaptées ou effectuer des combinaisons pour mieux analyser et comprendre des données.

Par exemple, il existe actuellement une approche (Henry *et al.*, 2007) qui a trouvé des avantages à combiner des matrices et des diagrammes noeuds-liens. D'une part, les matrices aident à montrer des réseaux denses et contournent le problème de chevauchement des liens et, d'autre part, les diagrammes noeuds-liens facilitent la compréhension de la direction des liens (Ghoniem *et al.*, 2005). Cette nouvelle approche a par la suite apporté des bénéfices pour la visua-

1. Source : http://www.emc.com/about/news/press/2011/20110628-01.htm.

lisation de communautés sociales. Il manque toutefois d'autres explorations de ce type afin de faciliter la compréhension de certains réseaux, dont les graphes composés et les graphes dynamiques.

Un autre problème est la complexité de ces données et il existe également des compromis dans le choix des moyens utilisés pour la gérer. En effet, l'application de ces techniques peut nuire à la compréhension de phénomènes sous-jacents. Par exemple, l'agrégation de données permet de réduire la complexité visuelle. Ainsi, cela peut aider à interpréter des informations à un plus haut niveau d'abstraction. Toutefois, en cachant des détails, peut-être critiques, cela peut également nuire à une compréhension plus avancée de ces mêmes données. Les réseaux sont typiquement grands et denses en pratique, ce qui entraîne également certaines complications. En particulier, les superpositions de liens et de noeuds peuvent empêcher de bien voir d'autres noeuds et relations potentiellement plus importants. Somme toute, la capacité d'analyse des informations dépend de la disponibilité des données, de la méthode visuelle utilisée et des possibilités d'interaction et d'exploration.

Ces défis demeurent pour des applications plus pratiques, par exemple en visualisation de logiciels (Diehl, 2007). En effet, les logiciels peuvent être modélisés par des graphes (les noeuds sont des modules) et la visualisation a le potentiel de pouvoir faciliter la compréhension de conceptions, souvent modifiées au fil du temps. En plus, l'analyse et le traçage de conceptions en évolution sont des défis importants dans ce domaine, alors que les approches visuelles sont encore émergentes (Gallagher *et al.*, 2008; Mens *et al.*, 2005). D'une part, les logiciels qui sont entreposés dans des répertoires de code source peuvent évoluer sur une longue période de temps et sont écrits dans des langages variés. D'autre part, l'interprétation d'architectures logicielles est complexe et nécessite des connaissances avancées en génie logiciel. Ainsi, en plus de savoir reconnaître des éléments de conception, un utilisateur doit être en mesure d'en évaluer les effets, parfois combinés, sur la qualité d'un logiciel en évolution au cours du temps. En particulier, les structures de conceptions, les patrons et anti-patrons sont des exemples d'éléments qui peuvent se retrouver dans une architecture à certains moments, rendant plus difficile l'évaluation des conceptions au fur et à mesure qu'elles se transforment.

L'objectif général de cette thèse est d'explorer et de valider des nouvelles approches afin de faciliter la compréhension de réseaux et de logiciels qui peuvent aussi évoluer dans le temps. Pour ce faire, des espaces de conception de techniques visuelles ont été explorés, dans le but de mieux appréhender la complexité de réseaux. Dans ce contexte, des techniques telles que l'agrégation et la combinaison d'approches ont été considérées. Afin de mieux documenter les possibilités de visualisations, nous avons conçu des taxonomies illustrant comment combiner des approches existantes. Certaines des options les plus prometteuses ont par la suite été implémentées dans des prototypes. De plus, nous avons tenté de renforcer nos approches par des interactions et d'autres fonctionnalités pour aider à répondre à des questions concernant des structures de données et leur évolution.

Les trois principaux chapitres de cette thèse traitent de différentes perspectives de la visualisation de structures complexes dans des réseaux statiques et dynamiques. Ainsi, le premier chapitre se concentre sur la combinaison de représentations pour des réseaux comportant des hiérarchies. Dans ce cas, des combinaisons de matrices, de diagrammes noeuds-liens et d'agrégations interactives aident à réduire la complexité visuelle et faciliter la compréhension de réseaux. Le deuxième chapitre explore aussi des combinaisons de visualisations, mais pour des graphes dynamiques. Enfin, le troisième chapitre décrit une nouvelle méthode visuelle et interactive afin de mieux appréhender l'évolution de structures de conception dans des logiciels (modélisés par des réseaux). Nos contributions générales sont (1) la documentation de nouvelles possibilités en termes de combinaisons de techniques de visualisation, (2) l'exploration et la validation de certaines de ces combinaisons en pratique pour analyser des réseaux, (3) une nouvelle méthode visuelle pour faciliter le traçage de structures de conceptions en évolution dans le temps.

Les nouvelles approches présentées dans cette thèse ont été validées expérimentalement (quantitativement et qualitativement) avec des participants humains. Plus spécifiquement, dans les deux premiers chapitres, les participants ont effectué des tâches avec et sans des combinaisons visuelles afin d'évaluer si celles-ci ont eu des effets sur les performances ou ont apporté d'autres bénéfices. Afin de valider les contributions du troisième chapitre, nous avons recruté

des participants avec des expériences en génie logiciel. Pour illustrer l'utilité pratique de nos interfaces dans ce contexte, nous avons exploré des projets réels, dans le but de découvrir des exemples de cas intéressants. Les participants à notre étude ont aussi analysé des historiques de logiciels en tentant de déceler, par exemple, des périodes de temps significatives, des zones d'instabilité ou d'autres patrons visuels clés.

Une autre stratégie pour atteindre nos objectifs a été de nous limiter, au niveau de la collecte des données, à (1) des projets de code source ouvert, (2) écrits en Java et (3) répertoriés dans Subversion [2]. De plus, une plateforme de code source réutilisable a été conçue afin de faciliter le développement des prototypes.

La suite de cette thèse est organisée comme suit. Dans le prochain chapitre, nous présenterons l'état de l'art en visualisation de réseaux et de logiciels dans le contexte des contributions principales visées, dans le but de voir quelles approches sont actuellement disponibles dans la littérature pour résoudre les problèmes qui viennent d'être discutés. Pour communiquer les contributions principales de la thèse, les trois articles de revues principaux intégrés à cette thèse seront ensuite présentés dans les chapitres subséquents, du plus général au plus spécifique. Ainsi, les deux premiers chapitres portent sur des visualisations visant essentiellement la communauté *infovis* et applicables à plusieurs domaines, alors que le troisième chapitre présente aussi une nouvelle approche visuelle, mais avec une portée ciblée sur le domaine de la conception de logiciels.

Le premier chapitre portera sur une nouvelle technique de visualisation de réseaux statiques afin de montrer les liens à divers niveaux de détails dans un graphe hiérarchique (TreeMatrix). Le second chapitre présentera une première approche hybride afin de montrer l'évolution de réseaux (DiffAni). Le troisième chapitre décrira un système de visualisation de structures de conception en évolution dans le temps et appliqué dans le domaine du génie logiciel (IHVis). Ensuite, en conclusion, nous discuterons de nos résultats et expliquerons comment nos contributions répondent aux problèmes soulevés, en comparaison avec les autres alternatives dans la littérature. Enfin, nous présenterons des idées de directions pour de futurs travaux.

2. Subversion est un logiciel de gestion des versions et est disponible à la page http://subversion.apache.org.

REVUE DE LITTÉRATURE

Les visualisations existent sous différentes formes, répondent à des besoins variés et sont applicables pour certains types de données et contextes. Cette correspondance fondamentale entre les structures de données et les techniques de visualisation est illustrée dans le cadre de référence (voir Figure 0.1). Selon ce modèle, la visualisation est une association entre les données et les formes visuelles utilisées afin de représenter ces informations. L'utilisateur peut interagir à chaque étape afin de contrôler ce processus de transformation des données.

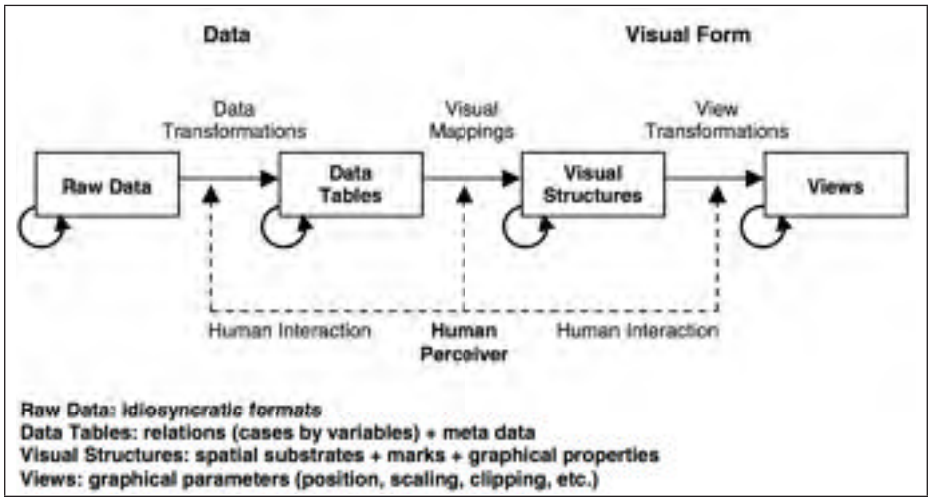

Figure 0.1 Modèle de référence général de visualisation (Card *et al.*, 1999).

Des approches de visualisation peuvent représenter des hiérarchies (Johnson et Shneiderman (1991); Schulz (2011); McGuffin et Robert (2010)), des réseaux (Herman *et al.* (2000); von Landesberger *et al.* (2010, 2011)), des données spatio-temporelles (Eccles *et al.* (2007); Andrienko *et al.* (2003)) ou multidimensionnelles (Hartigan (1975); Inselberg (2009); Mackinlay *et al.* (2007)). D'autres types de données et applications peuvent aussi nécessiter la création de visualisations spécialisées. En particulier, certaines structures de données sont la combinaison d'un ou plusieurs de ces types. Par exemple, les réseaux hiérarchiques (aussi appelés graphes composés) peuvent servir à montrer des données où les relations entre des éléments sont importantes, mais qui peuvent aussi être organisées en arborescence. Des exemples de ces différentes approches sont montrés dans la Figure 0.2.

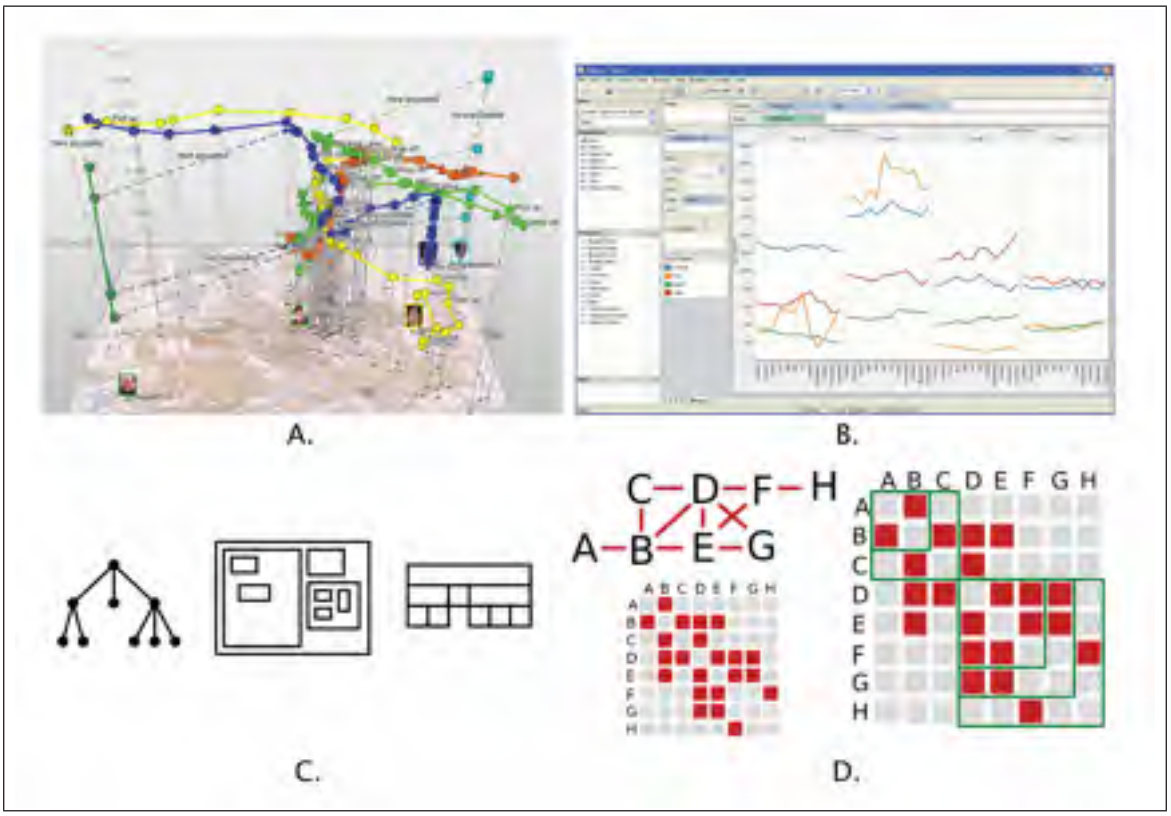

Figure 0.2 Exemples de visualisations de diverses structures de données. **A.** Données spatio-temporelles dans GeoTime (Eccles *et al.*, 2007), montrant des déplacements d'individus dans le temps (avec des animations) et dans l'espace. **B.** Tableau (Mackinlay *et al.*, 2007) permet de visualiser les tendances de données multidimensionnelles. **C.** Une hiérarchie est montrée de trois manières (arborescence en noeuds-liens, conteneurs, diagramme en glaçons). **D.** À gauche : Un réseau est illustré par un diagramme noeuds-liens et une matrice. À droite : une visualisation pour un réseau composé de hiérarchies.

La diversité des données fait en sorte que des approches visuelles peuvent ne pas être optimales dans certains contextes, et cette problématique a donné lieu à l'émergence de nouvelles approches et aussi de techniques hybrides. Dans le reste de ce chapitre, nous faisons le survol des visualisations de réseaux statiques, dynamiques et hybrides. Par la suite, l'état de l'art concernant des applications de certaines de ces visualisations en génie logiciel, qui peuvent être modélisés par des graphes, sera présenté. Enfin, nous survolerons les méthodes d'évaluation des interfaces en visualisation.

Visualisations de réseaux

Un réseau (aussi appelé graphe) est une structure de données répandue qui peut modéliser des relations entre des objets issus de divers domaines, tels que les réseaux sociaux, les réseaux de communications, le transport, les interactions biologiques ou les transactions financières (voir Figure 0.3).

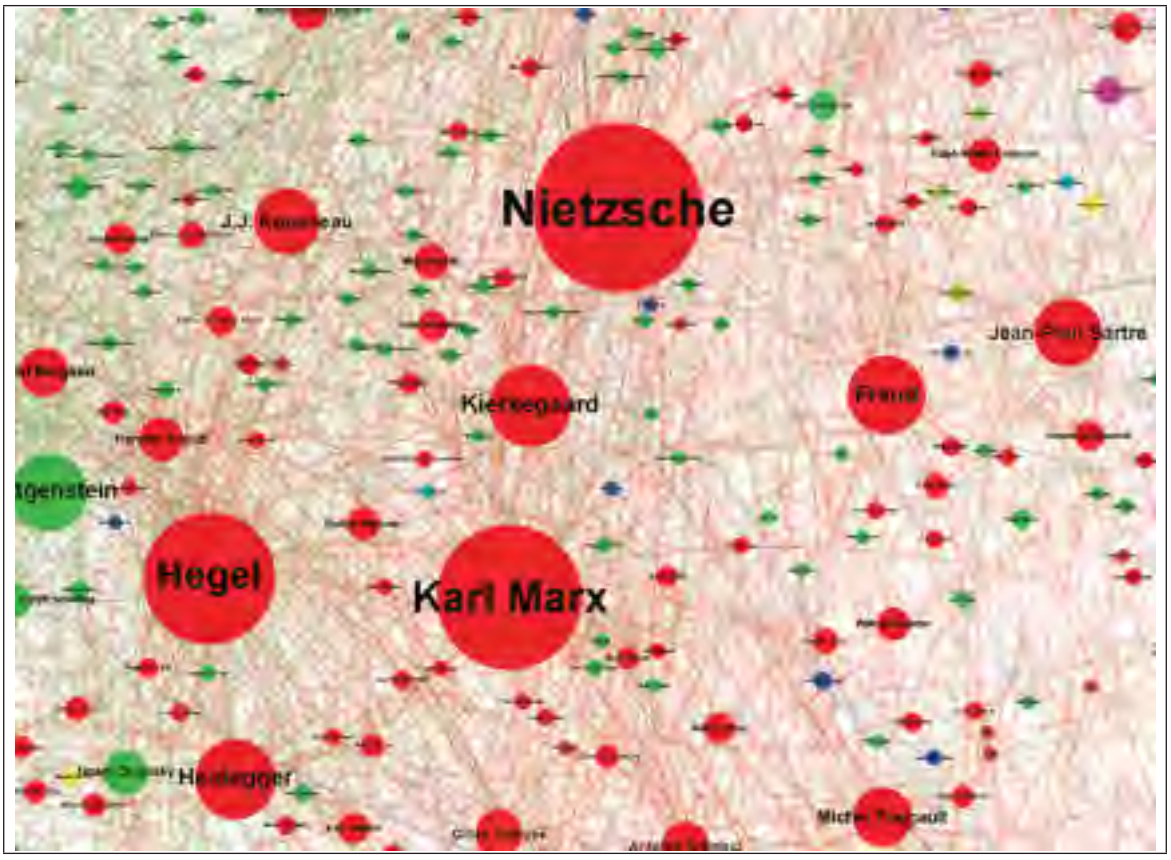

Figure 0.3 Réseau social montrant l'influence de penseurs sur le partage des idées (tiré de http://griffsgraphs.com/2012/07/03/graphing-every-idea-in-history/). La taille des noeuds indique le degré de l'influence et les échanges entre les personnes sont montrés par les liens.

Chaque graphe comporte un ensemble de noeuds qui sont interconnectés par des liens, qui peuvent être dirigés et pondérés, selon les cas (voir Figure 0.4).

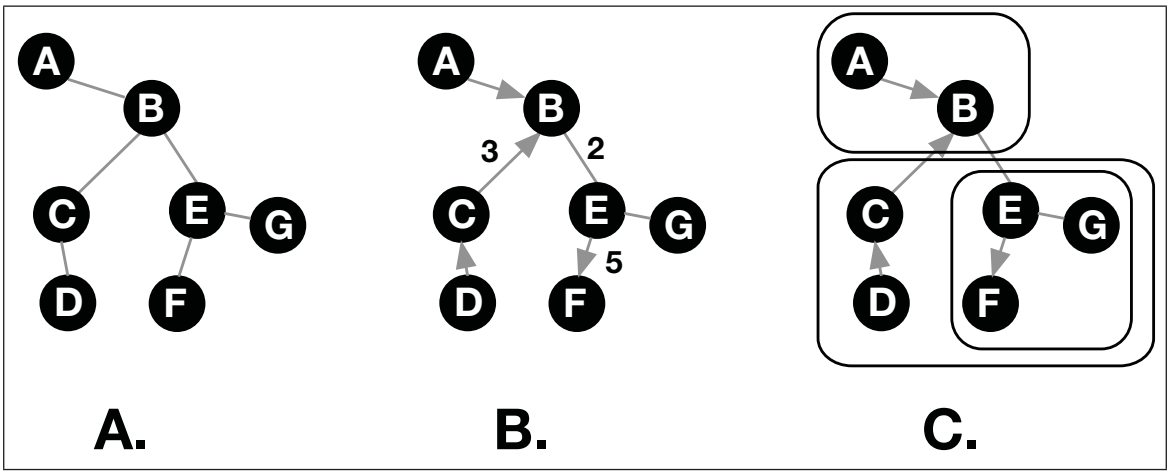

Figure 0.4 Sortes de graphes. **A.** Graphe sans liens pondérés ni dirigés. **B.** Graphe avec des liens pondérés et dirigés (si non précisé, les liens ont une pondération de 1). **C.** Graphe hiérarchique avec des liens dirigés. Ainsi, les noeuds AB et CDEFG forment des groupes parents, et ce dernier contient aussi un sous-groupe de noeuds enfants (EFG).

Par exemple, lors de l'analyse d'un réseau social, les degrés des liens entre les individus d'un groupe de personnes peuvent ne pas nous intéresser si notre objectif est surtout d'identifier les grandes communautés. Certains graphes peuvent également être structurés en hiérarchies (aussi appelés graphes composés). Dans le domaine du génie logiciel, par exemple, on peut modéliser les conceptions par des réseaux dirigés, pondérés et composés. Ainsi, une visualisation supportant ces types de graphes pourrait représenter les liens forts de modules qui dépendent d'une librairie graphique pour bien fonctionner, d'une part, et la hiérarchie de modules composant le système dans son ensemble, d'autre part.

Dans la littérature, les chercheurs utilisent divers algorithmes afin de dessiner les graphes (Di Battista *et al.*, 1999; Herman *et al.*, 2000; Kaufmann et Wagner, 2001; McGuffin, 2012) ou des graphes composés (Sugiyama et Misue, 1991; Sander, 96; Bertault et Miller, 1999). Les travaux antérieurs pour visualiser des graphes non hiérarchiques sont surtout basés sur des matrices ou des diagrammes noeuds-liens. Les matrices peuvent supporter des graphes denses et n'ont pas le problème du chevauchement des liens, contrairement aux diagrammes noeuds-liens, mais occupent beaucoup d'espace, d'où des travaux récents sur la compression de cet espace pour certaines données spécifiques (Dinkla *et al.*, 2012). Aussi, l'ordonnancement des

rangées et des colonnes des matrices est importante (Henry et Fekete, 2006; Elmqvist *et al.*, 2008; Mueller *et al.*, 2007) pour faciliter la découverte de regroupements potentiellement intéressants. Le diagramme noeuds-liens est populaire, et semble faciliter la compréhension de chemins (Ghoniem *et al.*, 2005), mais il existe une variété d'algorithmes de disposition des noeuds (Fruchterman et Reingold, 1991; Sugiyama *et al.*, 1981b), chacun avec leurs limitations potentielles.

D'autres approches supportent la visualisation de graphes hiérarchiques ("clustered graphs" ou "compound graphs"), bien qu'elles sont plus rares (von Landesberger *et al.*, 2011). Les hiérarchies d'un graphe composé peuvent venir des données, ou bien déduites selon des heuristiques ou des algorithmes (Gansner *et al.*, 2005; Archambault *et al.*, 2009; Elmqvist et Fekete, 2010), dans le but de mieux comprendre les données à haut niveau en cachant des détails (Balzer et Deussen, 2007; Gansner *et al.*, 2005).

La Figure 0.5 présente les techniques utilisées dans la littérature pour visualiser les graphes composés.

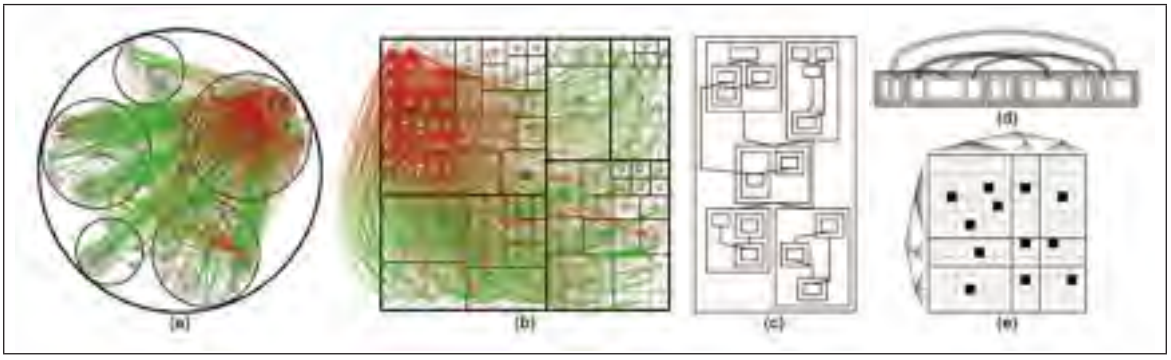

Figure 0.5 Approches pour visualiser des graphes hiérarchiques (Holten, 2006). **(a)** Les regroupements des noeuds sont indiqués par des bulles dans une disposition circulaire. **(b)** Des liens sont tracés par-dessus un *treemap* (Fekete *et al.*, 2003). **(c)** L'arborescence des noeuds est montrée par l'encapsulation des noeuds. **(d)** ArcTrees (Neumann *et al.*, 2005) montre les liens entre des éléments disposés horizontalement dans des conteneurs. **(e)** Matrice contenant des hiérarchies (Sangal *et al.*, 2005; Henry et Fekete, 2007; Rufiange *et al.*, 2009).

Les approches de visualisation de graphes composés illustrées dans la Figure 0.5 ont chacune leurs avantages et inconvénients. Ainsi, les liens sont plus difficiles à distinguer dans **(a)** et **(b)**. De plus, la matrice **(e)** rend le repérage de liens moins intuitif qu'avec **(c)** et **(d)**, mais ces dernières visualisations fonctionnent moins bien avec de larges réseaux et de profondes arborescences.

Les techniques vues jusqu'à présent permettent de montrer l'état d'un réseau à un moment fixé dans le temps. Néanmoins, un certain nombre de domaines nécessitent également la modélisation de l'évolution de graphes dynamiques. Pour ce faire, il existe dans la littérature des approches statiques ("Small Multiples"), dynamiques (animation) ou de comparaison (cartes de différences). Un graphe qui évolue dans le temps peut subir des changements en termes de noeuds (attributs, ajouts ou suppressions), de liens (degrés, ajouts ou suppressions), de positions des noeuds (si déterminées par la topologie du graphe, comme c'est le cas avec les dispositions basées sur les forces (Fruchterman et Reingold, 1991)).

D'abord, pour des changements concernant les attributs seulement, des "time series" (ou glyphs graphiques montrant l'évolution de mesures) peuvent être simplement tracées à l'intérieur de l'espace alloué pour les noeuds (Saraiya *et al.*, 2005b). Toutefois, pour montrer l'évolution de changements structurels, plusieurs "instantanés" du graphe peuvent être juxtaposés (aussi appelés "Small Multiples"). Une limitation est que beaucoup d'espace écran est requis et les utilisateurs peuvent prendre plus de temps à retrouver les noeuds qui se déplacent, à moins que des interactions ou des effets de surbrillance ne soient ajoutés pour compenser.

L'animation est une autre alternative (Frishman et Tal, 2008; North, 1996; Diel *et al.*, 2001; Erten *et al.*, 2004; Kumar et Garland, 2006). Dans ce cas, pour faciliter le traçage des changements au fil du temps, la carte mentale doit être préservée. Pour ce faire, les positions des noeuds doivent changer le moins possible au cours de l'historique d'un graphe dynamique. Néanmoins, dans la littérature, les études empiriques n'ont pas donné de résultats clairs concernant les effets de la carte mentale ou bien le degré de stabilité souhaitable (Archambault *et al.*, 2011a; Purchase et Samra, 2008; Purchase *et al.*, 2006). D'autres approchent utilisent le 3D (Brandes et Corman, 2003) ou l'interaction (Bezerianos *et al.*, 2010) pour montrer l'évolu-

tion dans le temps. Les cartes de différences (Archambault, 2009) sont similaires aux “Small Multiples”, mais indiquent en surbrillance les changements entre deux états (possiblement non consécutifs) d’un graphe dynamique. Lorsqu’une visualisation implique plusieurs (plus de deux) états distincts en même temps, il s’agit plutôt de vues multiples, qui seront discutées dans la prochaine sous-section (page 15).

Des approches permettent aussi de visualiser l’évolution de graphes composés (Burch et Diehl, 2008; Greilich *et al.*, 2009). Par exemple, TimeRadarTrees (Burch et Diehl, 2008) représente les noeuds d’un graphe à l’aide de secteurs. Ces secteurs sont d’abord fractionnés afin de montrer les liens entrants et sortants des noeuds. Chaque secteur est ensuite subdivisé en tranches pour illustrer l’évolution des noeuds (tel que le degré de modifications apportées) et des liens. Ainsi, les changements en termes des liens donnent lieu à des fractionnements différents des secteurs au fil du temps. Néanmoins, la hiérarchie est montrée de la manière classique, et cette technique semble moins appropriée lorsque l’arborescence des noeuds change fréquemment et significativement. Ces techniques sont intéressantes, mais sont moins efficaces pour des réseaux comportant de profondes hiérarchies ou bien pour des grands réseaux. Il manque aussi des évaluations avec des participants pour ces visualisations, étant donné leur plus grande complexité.

Visualisations hybrides

Bien qu’une variété de méthodes peuvent représenter des structures de données distinctes, elles imposent aussi des contraintes, pouvant rendre plus difficiles ou inappropriées certaines applications et potentiellement nuire à la compréhension de données. Par exemple, les matrices peuvent supporter les grands réseaux contenant beaucoup de noeuds et liens, sans avoir de problèmes de chevauchement de liens. Toutefois, les diagrammes noeuds-liens sont plus familiers et permettent de mieux comprendre les interconnexions entre des éléments (Ghoniem *et al.*, 2005).

Une piste potentielle est la combinaison des avantages de ces approches par la création d’un hybride, ce qui fut exploré pour la première fois par NodeTrix (Henry *et al.*, 2007). Cette

nouvelle approche, illustrée dans la Figure 0.6, permet de montrer par exemple des relations entre des communautés. En outre, cette technique aide à voir que “Ben Shneiderman” (il s’agit du noeud sélectionné en bas à droite de la figure) est connecté à plusieurs chercheurs au sein de sa communauté proche (via une matrice), mais aussi ses relations avec les autres communautés (via un diagramme noeuds-liens).

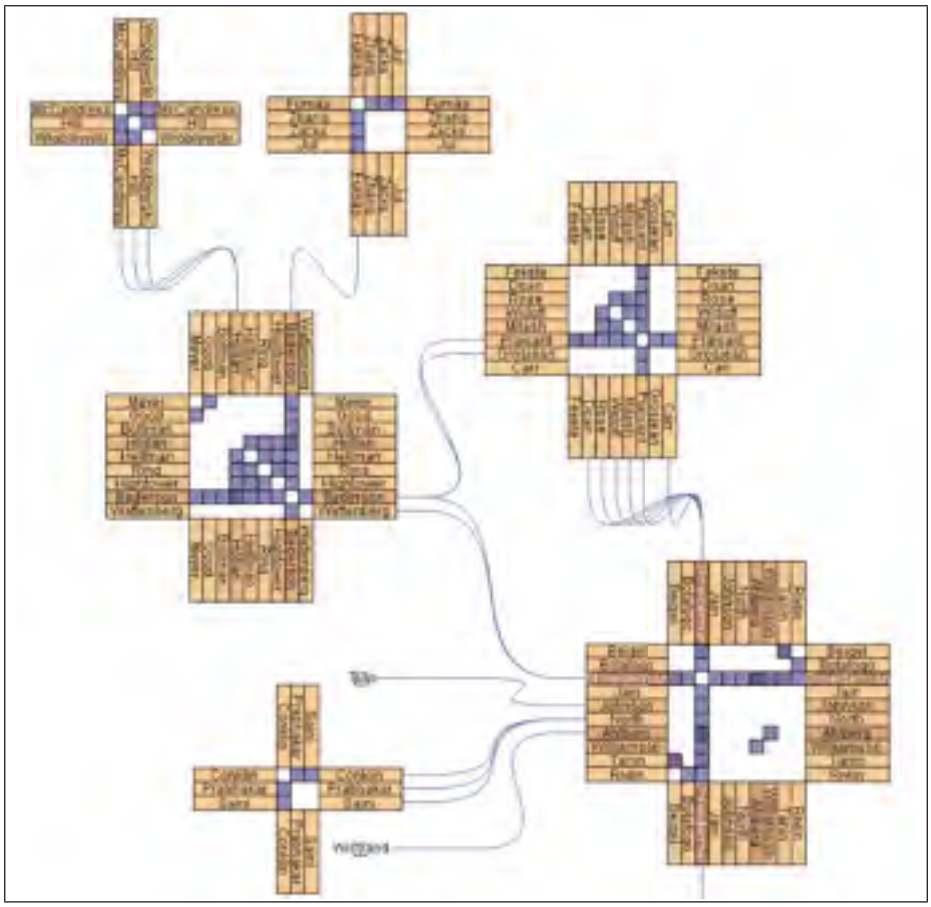

Figure 0.6 Aperçu de la visualisation hybride de graphes NodeTrix (Henry *et al.*, 2007). Des communautés de chercheurs (par exemple, des équipes de recherche dans une université) dans le domaine de la visualisation d’information sont montrées par des matrices d’adjacence, alors que les liens entre plusieurs communautés sont indiqués par des diagrammes noeuds-liens.

Une évaluation avec des participants (Henry *et al.*, 2008) a par la suite permis de valider des applications pratiques de cet hybride, comme l’identification de connexions entre des communau-

tés sociales. Néanmoins, ces nouvelles approches hybrides, potentiellement plus compliquées, sont rarement évaluées à l'aide d'expériences utilisateurs.

Il existe généralement peu de visualisations hybrides pour des données structurées en réseaux. Néanmoins, les "Onions Diagrams" (Sindre *et al.*, 1993) sont un autre exemple d'hybride pouvant montrer des graphes dans la littérature. D'autres hybrides peuvent aussi aider à montrer des hiérarchies (Elastic Hierarchies (Zhao *et al.*, 2005), Fekete *et al.* (2003)) ou des données multidimensionnelles (Viau *et al.*, 2010).

Il existe aussi peu d'explorations des espaces de conception des hybrides. Dans Elastic Hierarchies (Zhao *et al.*, 2005), une taxonomie (voir Figure 0.7) sert à illustrer les possibilités de combinaisons de matrices et de *treemaps* et leur prototype permet de tester plusieurs des options. Toutefois, il manque des travaux pour explorer l'utilité et les possibilités de combi-

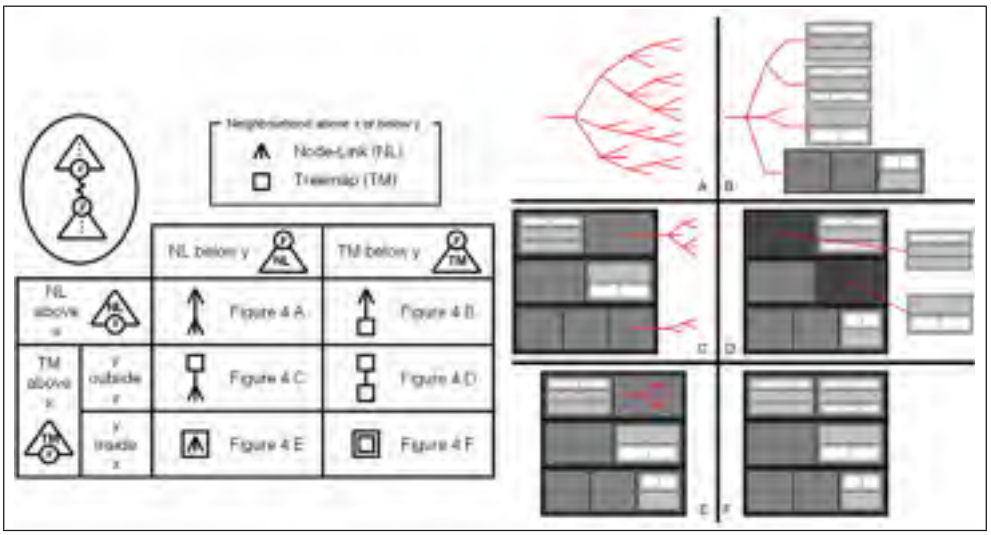

Figure 0.7 Exemple d'une taxonomie montrant des combinaisons possibles de visualisations de hiérarchies (des *treemaps (Johnson et Shneiderman, 1991)*) et de réseaux (noeuds-liens), tirée de (Zhao *et al.*, 2005). Chaque cellule de la taxonomie (à gauche) correspond à une possibilité de combinaison des deux approches, représentée à droite. Par exemple, dans le cas de la cellule "4C", un *treemap* montre la hiérarchie d'un réseau, et certains de ces noeuds sont plutôt montrés par un diagramme noeuds-liens à l'extérieur du *treemap*. La cellule du dessous, "4E" illustre un cas semblable, mais les nœuds-liens sont tracés à l'intérieur du *treemap*.

naison d'hybrides pour la visualisation de réseaux dynamiques, de même que la visualisation de graphes hiérarchiques. Un travail qui a exploré un hybride de visualisation de graphe dynamique est Hadlak *et al.* (2011), montré dans la Figure 0.8.

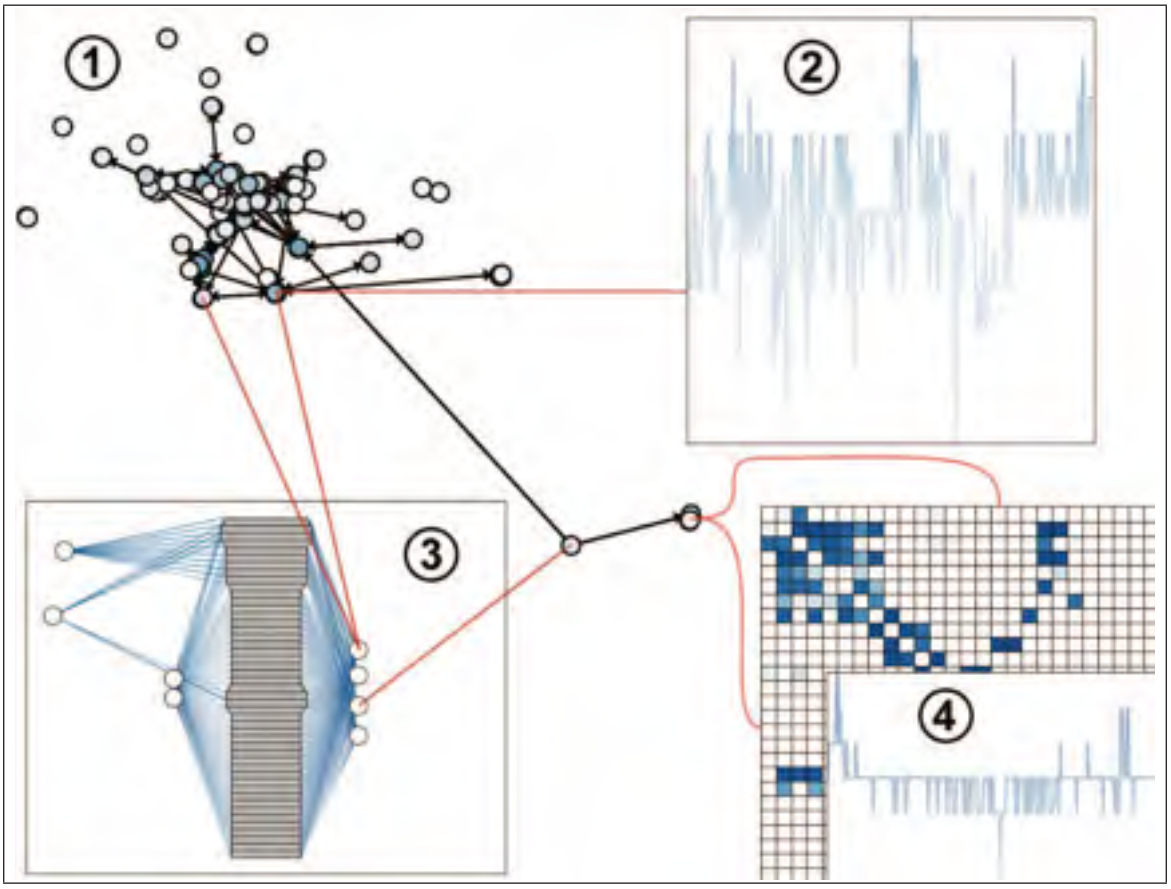

Figure 0.8 Exemple de visualisation hybride de graphe dynamique (Hadlak *et al.*, 2011). **1.** Un réseau est visualisé par un diagramme noeuds-liens classique, mais certaines parties (des sous-graphes) sont montrées de différentes manières (2,3,4). **2.** Un graphique montre l'évolution de la complexité au fil du temps du sous-graphe correspondant. **3.** Un sous-graphe est visualisé dans cette vue séparée, mais les liens avec le reste du système sont indiqués en rouge. **4.** Une partie d'un graphe est montré par une matrice d'adjacence. En plus, un sous-graphe dans cette partie de graphe est sélectionné et l'évolution de sa complexité est montrée par le graphique à base de lignes.

L'usage de vues coordonnées (Roberts, 2007) est une autre manière de montrer diverses informations concernant des visualisations. Par exemple, ces approches (Henry et Fekete, 2006; Zaman *et al.*, 2011) affichent deux vues juxtaposées et l'utilisateur peut interagir avec certains noeuds dans une vue afin de voir les correspondances dans la seconde. VisLink (Collins et Carpendale, 2007) permet de comparer des vues coordonnées en traçant les liens entre ces vues dans l'espace. Ces différents cas sont illustrés dans la Figure 0.9.

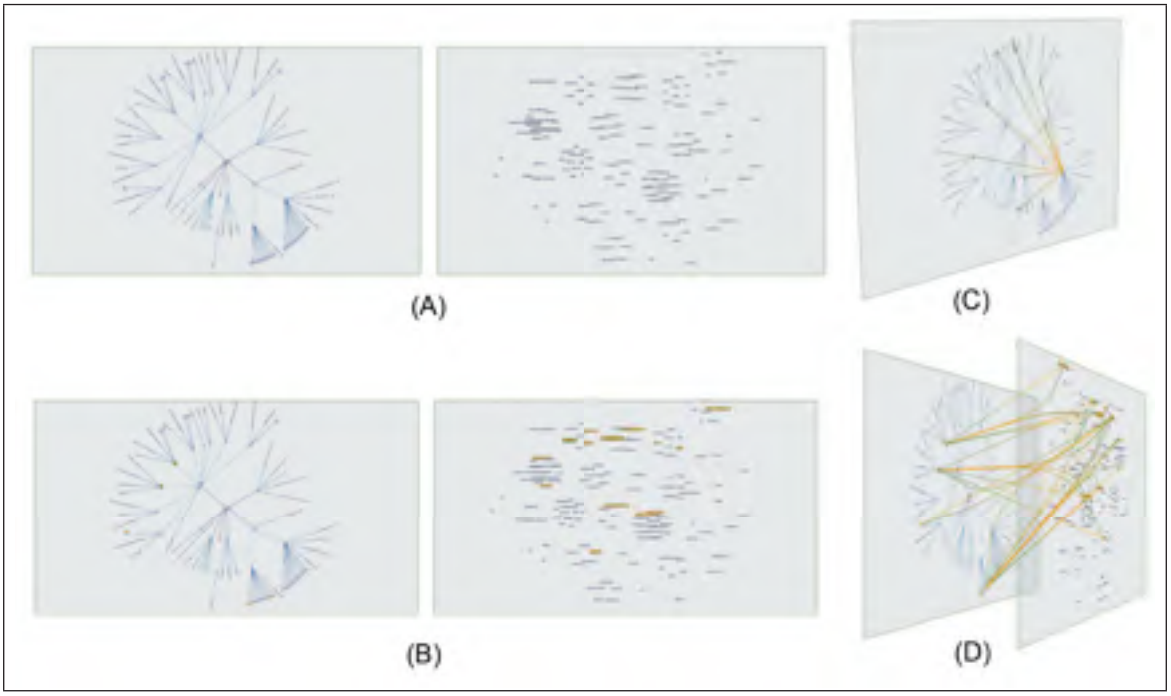

Figure 0.9 Différentes manières de visualiser des vues multiples (Collins et Carpendale, 2007). **A.** Vues juxtaposées (sans surbrillance). **B.** Vues coordonnées (avec surbrillance). **C.** Vues unifiées. Une première vue est dessinée en arrière-plan, et d'autres relations sont mises en surbrillance en superposant une seconde vue. **D.** VisLink. Les correspondances entre les vues sont reliées par des flèches dans l'espace.

Contrairement aux hybrides, les vues multiples ne permettent pas d'interchanger interactivement les représentations dans le but de bénéficier de chaque technique, et occupent inefficacement l'espace visuel. Ainsi, deux vues pourraient montrer le même graphe dynamique de façons différentes, par exemple, à l'aide d'une animation et de "small multiples". Bien que ces deux vues pourraient être plus appropriées pour des parties différentes du réseau en évolution,

l'utilisateur devrait basculer fréquemment entre ces vues, rendant plus difficile la compréhension des données. D'autres techniques, comme par exemple le traçage d'arcs au-dessus de matrices d'adjacence (Henry et Fekete, 2007) ou entre des noeuds organisés en hiérarchies (Fekete *et al.*, 2003), peuvent aussi aider à mieux voir les relations dans un réseau.

Un travail récent a proposé une classification générale de ces manières de combiner des visualisations, sans toutefois couvrir les spécificités propres aux réseaux (Javed et Elmqvist, 2012). Les visualisations peuvent ainsi être assemblées par juxtaposition (sans liens entre les vues, comme les vues coordonnées (Roberts, 2007)) ou par intégration (avec des liens entre les vues, comme VisLink (Collins et Carpendale, 2007)). Aussi, des visualisations peuvent être simplement superposées dans un même espace ou bien surchargées. Dans le cas de vues surchargées, une vue peut utiliser l'espace (ou la disposition) d'une autre vue (voir par exemple Fekete *et al.* (2003)). L'imbrication d'une vue dans une autre (comme NodeTrix (Henry *et al.*, 2007)) est une autre méthode de combinaison d'approches.

Dans ce cadre, notre approche hybride présentée au premier chapitre (TreeMatrix) inclut des vues imbriquées (découpage des données en plusieurs matrices et diagrammes noeuds-liens) et des vues surchargées (arcs et arborescences). Notre prototype décrit au deuxième chapitre (DiffAni) combine des représentations à l'aide de juxtapositions. Contrairement aux vues coordonnées traditionnelles, où chaque vue présente une perspective différente sur des données, chacune des vues dans DiffAni est plutôt associée à une période de temps distincte d'un graphe dynamique.

Visualisations de logiciels

L'adaptation des connaissances issues du domaine de la visualisation d'information au génie logiciel est récente. L'une des premières définitions de ce champ est la suivante : "Software visualisation is a discipline that makes use of various forms of imagery to provide insight and understanding and to reduce complexity of the existing software system under consideration" (Knight et Munro, 1999). Cette communauté s'intéresse plus particulièrement à visualiser la structure (statique), le comportement (l'aspect dynamique du code en exécution) et l'évolution

des logiciels, dans le but de mieux les comprendre et améliorer la productivité du processus de développement (survols récents dans Diehl (2007); Ghanam et Carpendale (2008); Caserta et Zendra (2011); Khan *et al.* (2012); Novais *et al.* (2013)). Concernant l'aspect dynamique, cet angle se concentre sur la visualisation de l'exécution de code en séquence, dans le but, par exemple, de comprendre un algorithme ou faciliter le débogage.

Les travaux en visualisation de logiciel couvrent ainsi plusieurs perspectives (statique, dynamique, évolution) et exploitent diverses techniques visuelles (2D, 3D, animation, basé sur des pixels, matrices, comparaison de différences) et dans le but de faciliter la compréhension d'un aspect (mesures, structure, relations, changements) à divers niveaux de détails (*packages* ou autres regroupements prédéfinis, classes, méthodes). Nous survolerons certains des travaux les plus représentatifs de ces efforts dans le domaine de la visualisation de logiciels.

Visualisations statiques de logiciels

La visualisation des aspects statiques d'un logiciel est utile pour mieux comprendre la structure du code et des relations. Dans la littérature, des diagrammes UML ou des graphiques statistiques, tels que des diagrammes à barres ou en radar (Kiviat), peuvent être générés (Ducasse *et al.*, 2005; Pinzger *et al.*, 2005). Par contre, ces approches relativement simples servent surtout à donner un aperçu général d'un projet, sans décrire les structures de manière détaillée (voir Figure 0.10). Ducasse *et al.* (2007) contourne ce problème en utilisant des "Blueprints". Ce travail se sert de l'organisation spatiale, sauf dans le cas de l'héritage, afin de montrer les relations entre les composants au lieu d'avoir recours à des liens explicites. Une approche semblable a été appliquée à des méthodes d'une classe (Lanza et Ducasse, 2001), alors regroupées en couches distinctes, selon leur nature (constructeur, interface, implémentation, accesseur, attribut). Toutefois, dans ce cas, les relations sont toujours montrées par des diagrammes noeuds-liens.

Lattix (Sangal *et al.*, 2005) a utilisé une matrice d'adjacence pour représenter les relations entre les composants d'un logiciel et la méthode n'est pas affectée par le chevauchement des liens. Une vue séparée montre l'organisation hiérarchique des éléments du système (voir Fi-

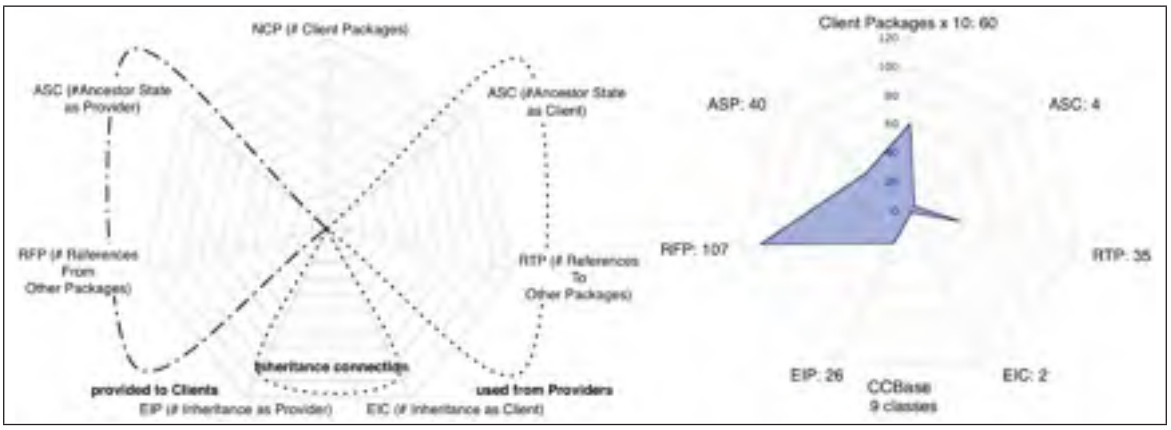

Figure 0.10 Visualisation statique à haut niveau d'un projet de code source (Ducasse *et al.*, 2005). Diverses métriques concernant les *packages* sont montrées en même temps en associant un axe à une mesure.

gure 0.11). Toutefois, les matrices rendent plus difficile l'interprétation de la direction des liens (Ghoniem *et al.*, 2005). Abdeen *et al.* (2010) ont aussi proposé une visualisation semblable basée sur une matrice pour voir les détails des relations à haut niveau entre les *packages* (taille, cohésion, couplages entrants et sortants).

Visualisations de l'évolution de logiciels

L'évolution des logiciels est un processus continu englobant plusieurs activités de la maintenance, en particulier les activités d'amélioration, d'adaptation et de correction, et débutant dès la livraison d'une première version (Ramil et Lehman, 2000). Le simple fait d'apporter des modifications contribue à la dégradation du logiciel (Parnas, 1994). Ce phénomène ne peut être contrôlé totalement, car "Our ability to design for change depends on our ability to predict the future" (Parnas, 1994). Néanmoins, la visualisation de l'évolution des logiciels est une approche afin de faire le suivi de ces changements constants. Bien qu'il s'agit d'un domaine récent, des taxonomies concernant les changements et l'évolution du logiciel (Mens *et al.*, 2003) ont été proposées (Buckley *et al.*, 2005). La représentation de l'évolution d'un logiciel étudie, d'une part, les mutations graduelles du logiciel au fil des changements et, d'autre part, les effets de ces modifications en général.

Figure 0.11 Lattix est une visualisation statique montrant les relations entre des modules, de même que leur organisation hiérarchique (Sangal *et al.*, 2005). Un diagramme à glaçons montre l'indentation de la hiérarchie. Ainsi, *print* et *help* sont des modules enfants de *jedit*.

L'évolution des données peut être visualisée en juxtaposant plusieurs représentations (SeeSoft (Eick *et al.*, 1992), SeeSys (Baker et Eick, 1995), CVSScan (Voinea *et al.*, 2005). En effet, CVSScan réutilise la représentation des lignes de SeeSoft, mais montre chaque fichier dans une colonne séparée. Alors que CVSScan montre l'évolution d'un fichier, CVSGrab (Voinea et Telea, 2006) permet de visualiser l'évolution d'un logiciel (de plusieurs fichiers). De plus, CVSGrab intègre les informations provenant de CVS et Bugzilla et permet de lancer des requêtes, par exemple afin d'étudier la propagation de changements de fichiers contenant des défauts (voir Figure 0.12).

D'Ambros et Lanza (2009) intègrent aussi les informations concernant les systèmes de gestion des versions et de suivi des modifications afin de donner une vue d'ensemble sur l'architecture, combinant divers aspects de l'évolution de ses composantes. Pour ce faire, ils font usage de

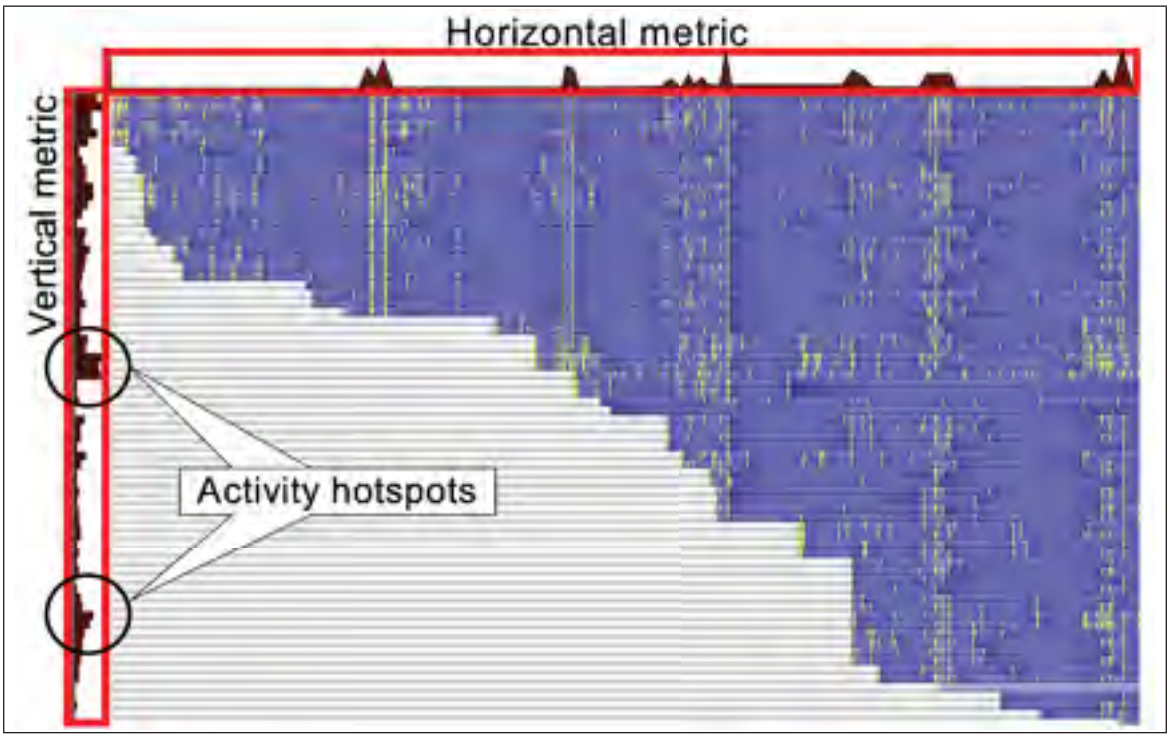

Figure 0.12 CVSGrab (Voinea et Telea, 2006) montre l'évolution de fichiers au fil du temps dans une vue compressée à base de pixels. Chaque ligne représente l'évolution d'un seul fichier et ces changements sont groupés par révisions dans un historique de code. À gauche, des mesures concernant les fichiers sont indiquées (activité, temps de création de fichiers, similitude avec un autre fichier sélectionné). En haut, des mesures du projet sont présentées graphiquement (par exemple, le nombre de fichiers changés à un moment dans le temps).

vues séparées (distributions des efforts de développement, hiérarchies de fichiers, graphes de dépendances). Cet outil ne permet pas de montrer l'évolution de lignes de code, contrairement à CVSGrab.

Chevalier *et al.* (2007) représentent aussi les fichiers par une technique semblable, mais le niveau de granularité est plus fin (au niveau des fonctions), utile notamment afin de détecter le "copier-coller", soit des blocs de code très similaires ("clone detection"). Chaque région est associée à une version d'un logiciel, sur un axe horizontal, et les liens entre celles-ci montrent l'évolution du code. Des diagrammes en glaçons montrent les hiérarchies à comparer (voir Figure 0.13). Telea et Auber (2008) ont proposé une approche semblable, et leur visualisation permet d'analyser la jointure ("Merge") ou séparation ("Split") de code entre des versions

successives et par la comparaison de hiérarchies. Aussi, Holten et Van Wijk (2008) ont proposé de comparer des hiérarchies de deux versions d'un code source en regroupant les modules en fonction de l'organisation hiérarchique du logiciel, et en traçant les interactions de manière à éviter les chevauchements.

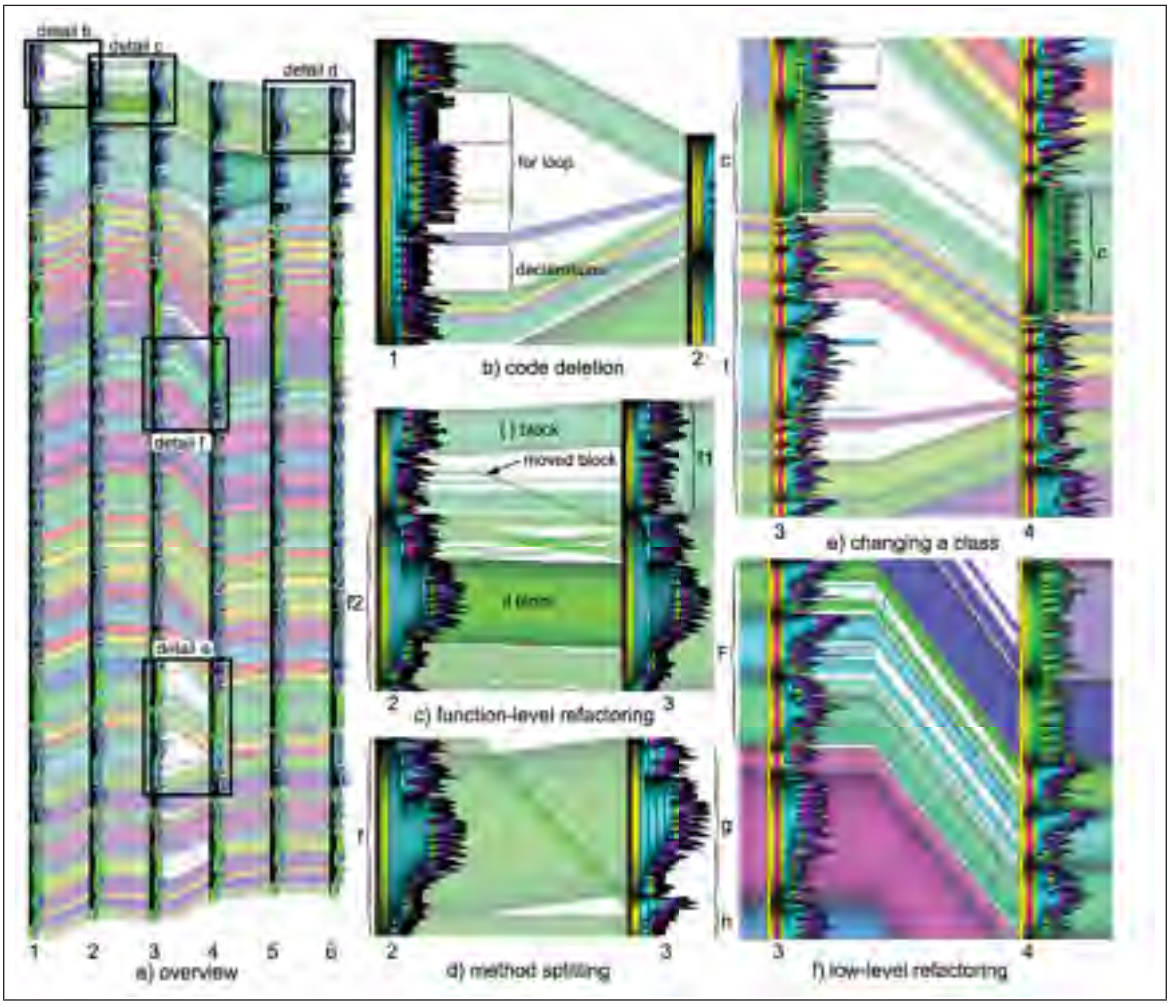

Figure 0.13 Chevalier *et al.* (2007) montre différents scénarios d'utilisation de leur visualisation afin de comparer deux versions de hiérarchies et constater les changements. Par exemple, dans le bas de la figure (*method splitting*), la méthode **(f)** a été restructurée en deux parties (**g** et **h**).

Revision Towers (Taylor et Munro, 2002) propose de visualiser une paire de fichiers (entête et implémentation en C/C++) en modifiant les propriétés visuelles de rectangles (longueur, largeur, couleur) selon diverses mesures (auteur, taille). Ceux-ci sont par la suite empilés les

uns sur les autres, en ordre chronologique. L'animation peut être utilisée afin de montrer le défilement des blocs, associés aux versions. Lanza (2001) dispose les versions d'un fichier, représentées aussi par des blocs, selon un axe temporel horizontal. Cette technique peut montrer l'évolution de plusieurs fichiers (un fichier par rangée).

SourceViewer3D (Xinrong *et al.*, 2006) visualise, à l'aide de diagrammes à barres en 3D, les données historiques à plusieurs niveaux de détails (système, fichier, fonction, ligne). Des couleurs indiquent des mesures du logiciel (taille des modifications, nombre de développeurs) ou de son évolution (nombre des transactions incluant un fichier ou la probabilité que deux fichiers soient couplés). Enfin, les graphiques de plusieurs versions d'un fichier sont placés les uns à côté des autres. Cette approche est semblable à Langelier *et al.* (2005), mais permet de montrer la force des relations entre des fichiers.

Weissgerber *et al.* (2007) présentent des informations concernant les développeurs à l'aide d'une matrice. En effet, les fichiers sont placés sur l'axe horizontal, alors que les auteurs sont inscrits sur l'axe vertical. Les noeuds sont rapprochés si les développeurs travaillent plus souvent ensemble. Burch *et al.* (2005) font également usage d'une visualisation semblable à une matrice (une carte de pixels) afin de visualiser des données concernant des règles d'association. Langelier *et al.* (2008) se servent de *treemaps* animés afin de montrer les changements dans les *packages* et les mesures (LOC, complexité, couplages), et ont trouvé des cas de transfert de responsabilité et de classes démesurées (voir Figure 0.14).

Un autre travail (Denier et Sahraoui, 2009) a permis de découvrir des observations concernant les hiérarchies (par exemple, la présence de grandes hiérarchies) à partir du code source. Par contre, leur technique ne peut pas représenter les implémentations d'interfaces multiples. Pour leur part, Girba *et al.* (2005) se concentrent sur la visualisation de changements impliquant des hiérarchies de classes à l'aide de diagrammes de classes semblables à UML. Hindle *et al.* (2007) représentent les relations entre des modules par un réseau, mais intègrent l'animation. Les liens peuvent être montrés durant toute l'animation dès que deux modules sont interdépendants (vue cumulative) ou seulement lorsque les relations évoluent. De plus, les liens changent de couleurs lors de modifications et sont redimensionnés selon l'âge du dernier changement.

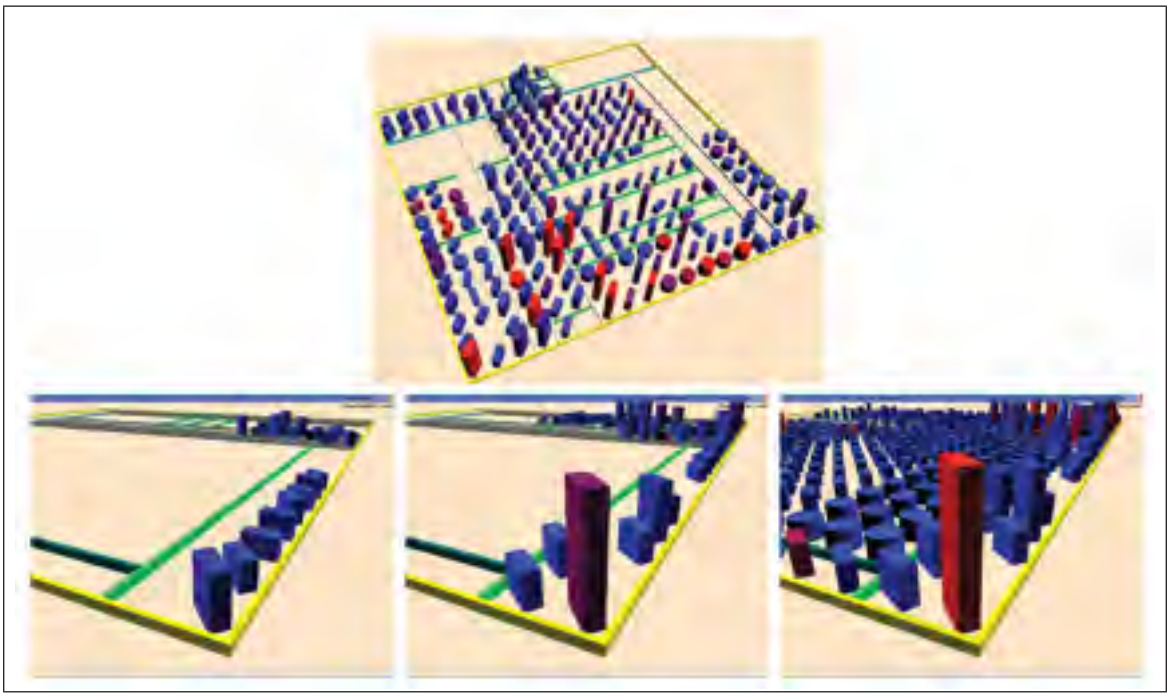

Figure 0.14 Visualisation proposée par Langelier *et al.* (2008) montrant la hiérarchie de classes (vue du haut). Dans la vue du bas, l'animation permet d'observer un problème potentiel de taille démesurée et croissante dans le logiciel libre *Freemind*.

Ces approches, qui montrent l'organisation en *packages* des logiciels, ne permettent pas de définir des logiques de regroupements personnalisés (par exemple, basé sur le couplage logique).

D'autres approches se concentrent sur la visualisation de l'évolution du couplage. Un radar, tel que proposé par D'Ambros *et al.* (2009), permet de visualiser les distances, en termes de couplage logique, entre un module et les autres classes du système. Les petits cercles représentent les classes du système, alors que celui-ci est divisé en secteurs (les modules), d'une manière similaire à Stasko et Zhang (2000). Deux fichiers peuvent sembler fortement couplés lorsque l'ensemble des données historiques est considéré, mais peut-être que cette relation était valable surtout pour la première année seulement. Afin de tenir compte de cette variation, D'Ambros *et al.* (2009) permettent à l'utilisateur de diviser l'historique en plusieurs périodes de temps. Par la suite, un radar et une mesure du couplage logique séparés sont associés à chacune de ces périodes.

EvoGraph offre une vue d'ensemble du couplage logique sur les classes d'un système (Fischer et Gall, 2006) par un graphique reliant par des lignes verticales les fichiers changés simultanément (voir Figure 0.15). Ratzinger *et al.* (2005) montrent aussi le couplage logique entre

Figure 0.15 EvoGraph (Fischer et Gall, 2006) aide à explorer le couplage logique avec deux vues : un aperçu général (diagramme noeuds-liens où les modules couplés sont plus proches les uns des autres) et une ligne de temps. Cette ligne de temps montre l'historique de changements et les changements simultanés (B,C) indiquent le couplage logique.

les classes, de même que les informations concernant les modules, mais ne permettent pas à l'utilisateur de se déplacer dans le temps. Beyer et Hassan (2006) proposent de disposer un

graphe selon l'énergie ("Energy-Based Layout"), dont les noeuds sont associés à des fichiers présents dans un répertoire CVS. La distance entre les noeuds est également liée au couplage logique. Diverses versions du graphe peuvent être juxtaposées ou l'animation peut servir à montrer l'évolution de ces fichiers. Dans ce cas, les noeuds sont colorés selon leur âge. Ainsi, cet outil fournit une meilleure vue d'ensemble de l'évolution que "Evolution Radar" (D'Ambros *et al.*, 2009), mais le discernement des liens structuraux et la lisibilité est compromise pour des réseaux denses.

L'usage de métaphore est une approche utilisée par certains chercheurs pour tenter de clarifier l'interprétation de données, alors que le code source et la conception d'un logiciel sont des notions intrinsèquement abstraites. Ainsi, un logiciel est parfois représenté par une ville (Wettel et Lanza, 2008a,b, 2007; Alam *et al.*, 2009), tel qu'illustré dans la Figure 0.16, un système solaire (Graham *et al.*, 2004; Ogawa et Ma, 2009) ou un paysage (Balzer *et al.*, 2004). L'efficacité de ces modèles, inspirés de métaphores abstraites ou réelles, devra cependant être évaluée et comparée dans le futur (Ghanam et Carpendale, 2008).

Ces approches (dont celle montrée dans la Figure 0.16) aident à tracer l'évolution de données, mais ne sont pas conçues pour suivre l'évolution du couplage entre les éléments. Aussi, la représentation de l'organisation hiérarchique du logiciel est utile, mais parfois limitée. En effet, au fil du temps, la cohésion entre les éléments peut changer, demandant de valider ou changer les regroupements indiqués dans la visualisation.

En résumé, il existe plusieurs visualisations qui couvrent diverses perspectives d'un logiciel et ces prototypes intègrent des techniques d'interaction et fonctionnalités variées. Bien que ces approches sont utiles pour des tâches et contextes spécifiques, elles comportent aussi des limitations. Les matrices, les diagrammes UML et en radar (Kiviat) sont des exemples de visualisations statiques utilisées afin d'étudier les relations entre des modules ou la structure d'un logiciel à un moment dans le temps. À l'aide de ces outils, les ingénieurs logiciels peuvent explorer ou réorganiser les regroupements des conceptions de logiciels, comparer des options d'organisations de modules (par juxtaposition des vues), identifier des couplages potentiellement déviants d'une architecture initiale ou tracer des exigences à des modules.

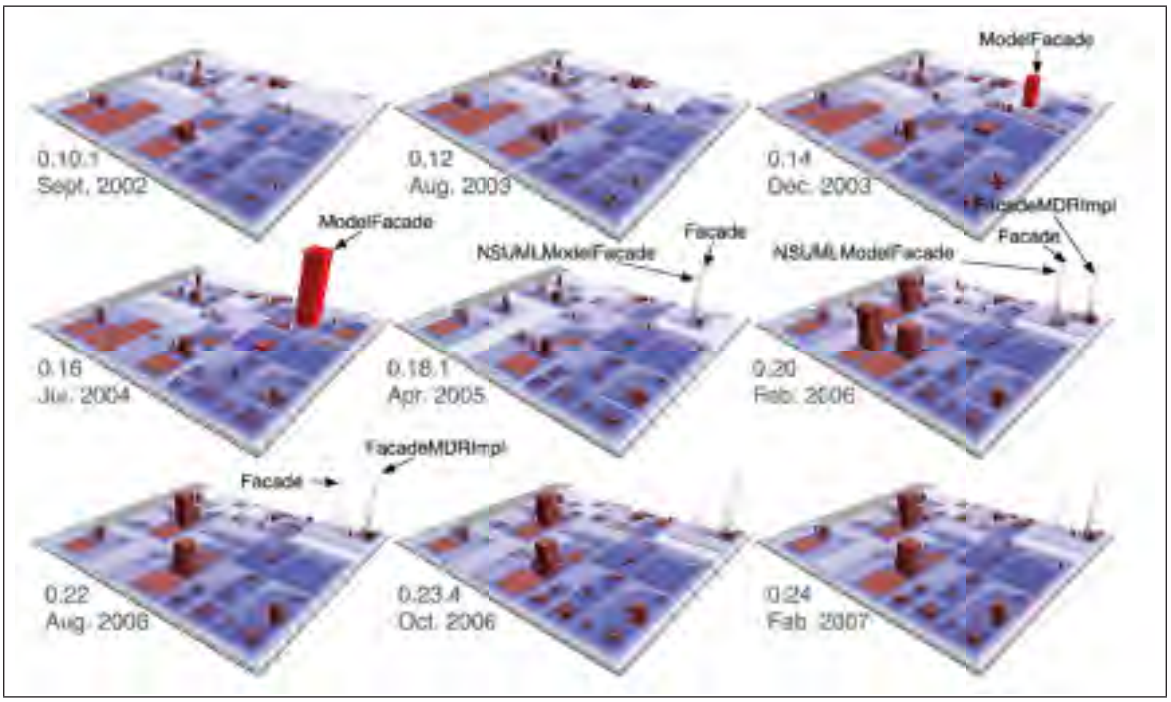

Figure 0.16 Métaphore de la ville représentant l'évolution (en termes de mesures de taille par exemple) du logiciel ArgoUML (Wettel et Lanza, 2008a). L'élément ModelFacade, en rouge, a augmenté en taille au fil du temps, puis a été décomposé en NSUMLModelFacade et Facade, probablement pour faciliter sa maintenance.

La visualisation de l'évolution du logiciel est un autre aspect important qui intéresse les ingénieurs afin de mieux comprendre le processus complexe d'évolution du code source. Afin de faciliter cette analyse, des chercheurs ont conçu des visualisations interactives qui intègrent des techniques telles que l'aggrégation de noeuds ou de liens, le zoom sémantique ou des algorithmes spécialisés de placement de noeuds. Les diagrammes noeuds-liens, les lignes de pixels, les *treemaps* et les matrices sont des exemples de représentations utilisées dans la littérature afin de montrer l'évolution de logiciels. Ces techniques peuvent aider à repérer des fluctuations en termes de taille, de complexité ou de couplage, des dégradations de qualité ou des restructurations. Malgré les avancées dans ce domaine très récent de la visualisation de logiciels, les chercheurs et ingénieurs ont besoin de plus d'approches pour analyser et tracer l'évolution de conceptions au fil du temps (Mens *et al.*, 2005; Gallagher *et al.*, 2008). Un autre problème est le manque d'études avec des participants afin d'évaluer et de comparer les interfaces visuelles proposées par les chercheurs.

Évaluations de visualisations

Le manque d'évaluations est un problème en visualisation d'information en général (Chen, 2005) et, en particulier, en visualisation de logiciels (Diehl, 2007; Caserta et Zendra, 2011; Novais *et al.*, 2013). En particulier, les nouvelles visualisations hybrides proposées dans la littérature sont rarement (sauf Henry *et al.* (2008)) évaluées avec des participants.

Il existe différentes manières de valider des interfaces avec des utilisateurs ou non, dépendamment des contributions et des domaines visés par les travaux. En visualisation d'information, l'objectif est habituellement de développer des techniques génériques pouvant servir à représenter des données issues de plusieurs domaines d'applications. Les nouvelles approches conçues sont souvent validées avec des expériences utilisateurs afin de vérifier si, par exemple, une approche améliore les performances ou réduit les taux d'erreurs pour certaines tâches.

Parmi les techniques de validation générales, mentionnons l'expérimentation contrôlée (surtout pour comparer des approches), l'évaluation de la convivialité (qualitative), les études de cas (voir Komlodi *et al.* (2004) pour un survol ou Kosara *et al.* (2003); Munzner (2009); Shneiderman et Plaisant (2006) pour des guides pour choisir la bonne approche de validation).

L'expérimentation contrôlée (évaluation quantitative) débute par des hypothèses de recherche à vérifier (par exemple, l'interface A est plus rapide qu'une interface B, l'animation aide à suivre les objets qui se déplacent, etc). Pour ce faire, les chercheurs manipulent des variables indépendantes en entrée (par exemple, une approche, tâche ou question) et mesurent l'effet en sortie (les variables dépendantes, telles que le temps pour effectuer une tâche, le taux d'erreur dans des choix multiples, le nombre d'essais, le temps passé à interagir avec des composants graphiques).

Les variables indépendantes peuvent être collectées de manière répétée ("within-subjects design") ou groupée ("between-group design" ou "between-subjects design"). Par exemple, dans le cas des mesures répétées, chaque participant essaie toutes les interfaces à comparer durant une même session, alors que pour les mesures groupées, les participants sont plutôt affectés à des groupes séparés au préalable et chaque groupe travaille avec une seule interface. L'ordre des

tâches et des ensembles de données peut être balancé en utilisant, par exemple, un carré latin afin d'éviter un biais dans les résultats. Par la suite, les données peuvent être collectées, puis analysées statistiquement. De cette manière, on peut évaluer si la moyenne des temps des participants pour une condition donnée (par exemple, l'interface A) est significativement meilleure qu'avec une interface B.

L'évaluation de la convivialité (qualitative) requiert moins de sujets et sert surtout à documenter les forces et faiblesses relevées par des utilisateurs et peut aider les concepteurs à améliorer leur produit. Elle peut aussi être complémentaire à l'expérience contrôlée. La méthode et la nature des questions peuvent varier selon les domaines (voir Stasko (2000); Sutcliffe *et al.* (2000); Marghescu *et al.* (2004) pour des exemples d'applications). Ces approches requièrent souvent des mesures moins précises, mais si elles sont quantifiées et objectives, cela peut fournir suffisamment d'informations pour suggérer de futures améliorations (Kosara *et al.*, 2003). Des expériences qualitatives peuvent aussi être effectuées (entrevues, sondages, observations, "focus groups").

L'étude de cas permet de démontrer la faisabilité et l'utilité d'approches, dans un environnement d'utilisation réel (Plaisant, 2004). Ces approches de validation sont répandues en sciences sociales (dont en psychologie) et permettent d'observer des situations dans des contextes précis (donc les résultats sont moins généralisables). Ces études peuvent fournir des indications pour des questions ciblées (Kosara *et al.*, 2003) et sont utiles si on ne sait pas encore clairement quelles hypothèses on veut vérifier, peut-être car les possibilités sont nombreuses. Les résultats obtenus suite à une étude de cas peuvent servir à trouver de nouvelles pistes de recherche prometteuses (possiblement non imaginées au départ) ou bien des hypothèses qui pourront être vérifiées par la suite (par exemple, via des expériences contrôlées).

Les applications concrètes en visualisation d'information, comme la visualisation de l'évolution de conceptions logicielles, sont surtout validées, dans la littérature, par des études de cas démontrant une utilisation possible des interfaces afin de retrouver des observations intéressantes. Il n'est pas simple de décider des tâches et de la méthodologie expérimentale à utiliser pour mettre en valeur une nouvelle approche. En outre, celles-ci peuvent nécessiter le recru-

tement de participants plus ou moins spécialisés. Les participants novices ou intermédiaires peuvent comparer la convivialité d'interfaces et ces résultats pourraient ainsi être plus généralisables à d'autres utilisateurs. D'un autre côté, certaines applications requièrent des connaissances avancées et les experts (qu'ils soient participants ou relecteurs) sont potentiellement plus en mesure d'évaluer la valeur concrète d'une nouvelle approche.

Les expériences utilisateurs sont également rarement utilisées pour valider ces nouvelles interfaces (un exemple est Ducasse et Lanza (2005)). En fait, les études de cas (ou "case studies"), qui consistent à expliquer des exemples de cas intéressants retrouvés dans un ou plusieurs projets de code source réels, sont clairement plus répandues. Une autre approche de validation connexe est basée sur les *insights*, c'est-à-dire "an individual observation about the data by a participant" (Saraiya *et al.*, 2005a).

Dans ce cas, les chercheurs (ou d'autres experts) peuvent *a priori* identifier des observations intéressantes et laisser les participants les trouver par eux-mêmes (North, 2006), soit en utilisant des choix multiples ou bien un champ de commentaires plus large afin de recueillir toutes leurs observations générales sur des ensembles de données distincts. Un problème potentiel est que des connaissances avancées peuvent être requises, ce qui peut rendre plus difficile le recrutement de participants. Autrement, les participants moins expérimentés peuvent ne pas être certains de ce qu'ils doivent chercher (ce qui pourrait révéler des observations inattendues, mais aussi augmenter la variabilité des résultats).

Plusieurs travaux antérieurs (publications dans des journaux) ont utilisé l'approche de validation par étude de cas. En particulier, les chercheurs présentent des cas intéressants (retrouvés dans un ou plusieurs ensembles de données) afin de valider leurs nouvelles interfaces, notamment en visualisation de logiciels (D'Ambros et Lanza, 2009; Hurtado Alegría *et al.*, 2013; Beck et Diehl, 2010; Vanya *et al.*, 2012; Abdeen *et al.*, 2010).

Les arguments des chercheurs (Beck et Diehl, 2010) illustrent bien la problématique de la validation d'interfaces dans ce contexte : "The case study will only show that the approach basically works. A task-oriented user study... may provide more reliable results on the usability,

readability, and efficiency of the proposed visualization technique. Nevertheless, it could be difficult to define an appropriate control group because there does not exist a directly competing approach."

Dans Vanya *et al.* (2012), ils ont effectué une étude de cas sur un projet et vérifié, par observations, comment les ingénieurs logiciels faisaient usage de leur visualisation pour réduire le nombre de couplages non souhaités. Ils expliquent aussi les limites de leur expérience : “With respect to internal validity : our study is not a controlled experiment. [...] There may be a bias in that we evaluate our own tool. The extensive feedback from the architect and developers, reflected in the quotes given, alleviates that to some extent”. Dans Abdeen *et al.* (2010), ils ont effectué une étude limitée avec quelques personnes pour valider leur nouvelle interface, mais concluent qu’une “real user study is required”.

Bien que la validation par études de cas peut demeurer plus qualitative qu’une expérience contrôlée, la réalité de ce domaine naissant de visualisation des logiciels est qu’on ne connaît pas encore bien toutes les hypothèses que l’on cherche à vérifier. De plus, en l’absence d’approches comparables, la comparaison formelle est moins applicable aux approches novatrices, qui n’ont pas encore d’alternatives. Dans cette optique, les études de cas ont le potentiel de pouvoir découvrir de nouvelles pistes de recherche encore inexplorées, et des possibilités d’hypothèses qui pourront être validées dans des expériences contrôlées dans de futurs travaux.

Bref, il existe plusieurs techniques afin de représenter des réseaux statiques, hiérarchiques ou dynamiques, dont les diagrammes noeuds-liens, les matrices et les cartes de différences. Ces approches sont utiles dans certains contextes, mais comportent aussi des limitations. En outre, les diagrammes noeuds-liens peuvent plus naturellement montrer les relations et les chemins entre des noeuds que des matrices, mais celles-ci supportent plus facilement les réseaux denses et contournent le problème de chevauchement des liens. D’une part, la combinaison d’approches (ou hybrides) est peu explorée pour la visualisation de certains types de réseaux. D’autre part, il manque d’interfaces visuelles pour analyser et tracer des aspects évolutifs des logiciels (modélisés par des réseaux).

Dans les prochains chapitres de cette thèse, nous explorerons les possibilités de combinaisons de techniques de visualisation de graphes à l'aide de taxonomies, puis décrirons des implémentations concrètes de certains hybrides sous la forme de prototypes. Ainsi, nous présenterons nos visualisations novatrices de réseaux hiérarchiques, de graphes dynamiques, ainsi qu'une nouvelle méthode visuelle d'exploration des variations au sein de conceptions logicielles.

CHAPITRE 1

ARTICLE 1. TREEMATRIX : A HYBRID VISUALIZATION OF COMPOUND GRAPHS

Sébastien Rufiange[1] and Michael J. McGuffin[1] and Christopher P. Fuhrman[1]
[1]Department of Software and IT Engineering, École de technologie supérieure,
1100, rue Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
sebastien@rufiange.com, {michael.mcguffin, christopher.fuhrman}@etsmtl.ca
Article soumis à la revue "Computer Graphics Forum" et accepté le 25 décembre 2011.
Présentation à EuroVis 2013 le 19 juin 2013 (parmi les deux articles invités à la conférence).

1.1 Abstract

We present a hybrid visualization technique for compound graphs (i.e., networks with a hierarchical clustering defined on the nodes) that combines the use of adjacency matrices, node-link and arc diagrams to show the graph, and also combines the use of nested inclusion and icicle diagrams to show the hierarchical clustering. The graph visualized with our technique may have edges that are weighted and/or directed. We first explore the design space of visualizations of compound graphs and present a taxonomy of hybrid visualization techniques. We then present our prototype, which allows clusters (i.e., subtrees) of nodes to be grouped into matrices or split apart using a radial menu. We also demonstrate how our prototype can be used in the software engineering domain, and compare it to the commercial matrix-based visualization tool Lattix using a qualitative user study.

Keywords : compound graphs, information visualization, taxonomy, software visualization, software design.

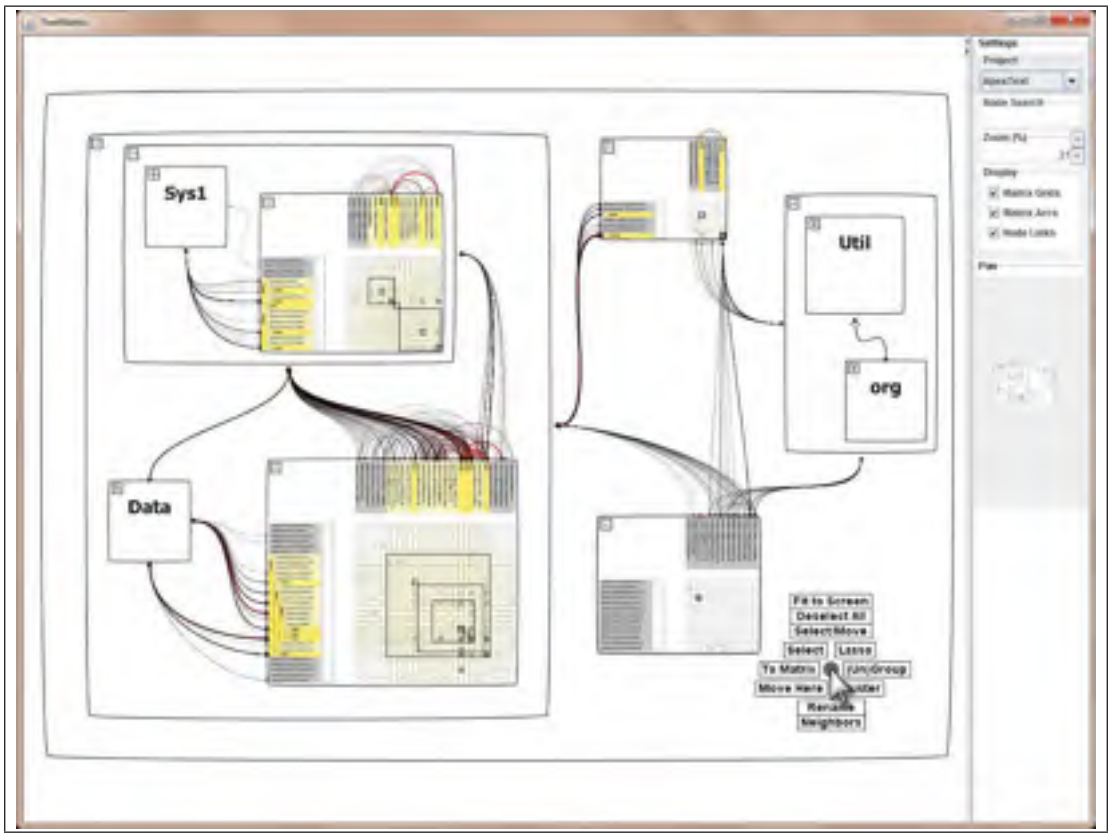

Figure 1.1 The TreeMatrix prototype displaying the structure of a set of source code files as a compound graph. Portions of the compound graph are shown as adjacency matrices. Some matrices have had their contents collapsed by the user, and only their names are visible (e.g., “Sys1”, “Util”). Rounded rectangles surrounding the matrices show the upper levels of the tree structure of the compound graph. Curves connecting the matrices show graph edges of the compound graph. At lower right, a radial menu is popped up to access commands.

1.2 Introduction

A compound graph, also called a clustered graph (Di Battista et Frati, 2009), is a graph (i.e., network) together with a rooted tree such that the leaves of the tree are the vertices of the graph. The non-leaf nodes of the tree can be thought of as meta-nodes or clusters of nodes. Compound graphs are useful for modelling situations where there is a network whose nodes can be grouped together into clusters. For example, a social network could be modeled as a compound graph, with graph edges representing relationships between people, and the tree clusters representing hierarchically-organized communities. As another example, in biology, a protein-protein inter-

action (PPI) network forms a graph whose nodes can be clustered hierarchically according to Gene Ontology (Gene Ontology Consortium, 2000) terms. Many algorithms exist for automatically computing a hierarchical clustering of a graph, essentially converting the graph into a compound graph.

Clusters are typically very meaningful to users in a specific application domain ; they help users understand the structure of the compound graph. These clusters, whether computed or provided with the input graph, can be used to aid visualization (e.g., Gansner *et al.* (2005); Balzer et Deussen (2007)) to limit the amount of detail the user sees at any given moment, e.g., by collapsing clusters of nodes and representing them as a single meta-node in the compound graph's tree. If the compound graph is large, such collapsing may be more convenient than trying to visualize all details of the graph at once. However, as pointed out in (Balzer et Deussen, 2007), a disadvantage of collapsing a cluster to a single meta-node is that detail is lost.

A second problem with many visualizations of graphs generally (not just compound graphs) is that of edge crossings obscuring information, especially in dense networks. To address this second problem, visualizations based on adjacency matrices (e.g., Bertin (1983); Sangal *et al.* (2005); Ghoniem *et al.* (2005)) have been proposed, which completely eliminate edge crossings. It is also possible to mix adjacency matrices with traditional node-link representations, as done with NodeTrix (Henry *et al.*, 2007). The current work presents a novel visualization for compound graphs that extends NodeTrix by combining multiple previous visualization techniques for trees and graphs. We call our visualization TreeMatrix (Figures 1.1, 1.2, 1.9), because it makes use of adjacency matrices for visualizing parts of a graph, and simultaneously displays the tree structure in a compound graph. The use of matrices addresses the problem of edge crossings mentioned above. It also helps mitigate the first problem mentioned above, that collapsing a cluster to a single meta-node can remove too much detail : instead of completely collapsing a cluster of nodes, the user may instead convert it to a matrix, to provide a visual summary of the nodes within it. As discussed later, our technique also combines aspects of MatLink (Henry et Fekete, 2007) and Lattix (Sangal *et al.*, 2005).

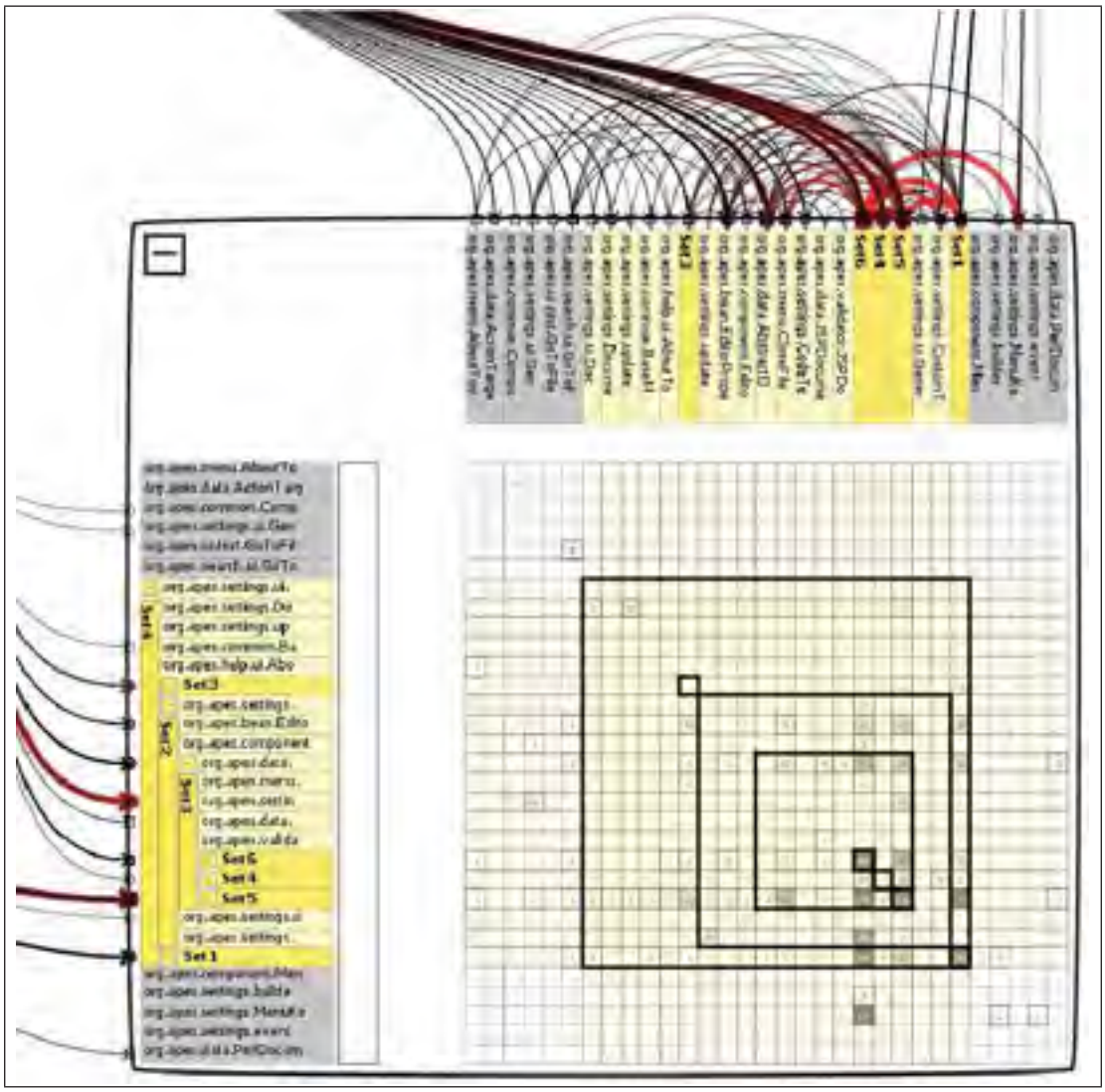

Figure 1.2 A zoomed view of the adjacency matrix in the lower left of Figure 1.1. Within each matrix of the visualization, the tree structure of the compound graph is shown with nested black squares inside the matrix, as well as with an "icicle diagram" listing nodes along the left edge of the matrix. Also, within each matrix, the weighted edges of the graph structure are shown by the numbers and colors in the cells of the matrix, as well as by 180-degree arcs along the top of the matrix.

Before presenting our visualization in detail, we first investigate the design spaces of hybrid visualizations of trees, hybrid visualizations of graphs, and visualizations of compound graphs. The taxonomies we present of these design spaces build upon the previously presented taxonomy in (Zhao *et al.*, 2005) and clarify the reasons behind our design choices. We then present our prototype implementation and demonstrate its use for visualizing, interpreting, and reverse

engineering the design of software source code. Our contributions are (1) new taxonomies of hybrid visualizations of trees, graphs, and compound graphs ; (2) a new visualization technique for compound graphs that extends and combines aspects of NodeTrix (Henry *et al.*, 2007), MatLink (Henry et Fekete, 2007), and Lattix (Sangal *et al.*, 2005) ; (3) a description of a software prototype implementing this technique, and (4) the results of a user study where our prototype was compared with Lattix, a status quo commercial product, for software design tasks.

1.3 Background

1.3.1 Visualization of Trees

Many techniques exist for visualizing trees (Jürgensmann et Schulz, 2010), including node-link diagrams, icicle diagrams, and nested enclosure approaches such as *treemaps* (Figure 1.3). As the depth of a tree increases, the number of nodes often increases exponentially, leading to crowding at the deeper levels. *Treemaps* (Johnson et Shneiderman, 1991; Bruls *et al.*, 2000;

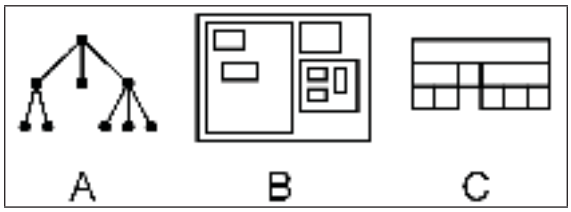

Figure 1.3 Three ways of drawing the same tree. **A :** node-link diagram. **B :** nested enclosure. **C :** icicle diagram.

Wattenberg, 2005) are a technique for drawing trees that make more efficient use of area than traditional node-link approaches, allowing more area to be allocated to each node. Treemaps are part of a more general category of tree representations which use *nested enclosure* to depict the tree structure. In an analysis of the space-efficiency of representations of trees (McGuffin et Robert, 2010), nested enclosure approaches (including treemaps) were ranked as asymptotically more efficient than node-link approaches. However, representations based on nested enclosure can also be more confusing than node-link diagrams. For example, judging the depth of a node within a treemap is more confusing than in a classical layered node-link diagram.

Hence, elastic hierarchies (Zhao *et al.*, 2005) were proposed, allowing users to visualize parts of a tree with treemaps, and other parts in node-link form.

1.3.2 Visualization of Graphs

Many layout algorithms have also been proposed for visualizing graphs (Di Battista *et al.*, 1999; Herman *et al.*, 2000; Kaufmann et Wagner, 2001), most of these based on node-link diagrams. Node-link diagrams, however, suffer from edge congestion when depicting graphs with many edges. An alternative to using node-link diagrams to depict graphs is to use adjacency matrices. Matrices have the advantage of completely eliminating edge crossings, but have the disadvantage of making path-following tasks more difficult (Ghoniem *et al.*, 2005). For this reason, enhancements have been proposed to matrices (Shen et Ma, 2007), including MatLink (Henry et Fekete, 2007) which adds arcs along the edge of the matrix. Arcs allow node adjacencies to be shown along a 1-dimensional layout, as previously shown in (Bertin, 1983) and applied to the visualization of repeating substrings in (Wattenberg, 2002).

Matrices also have a significant area cost ($\Theta(N^2)$), and so probably don't offer much advantage, if any, when the graph to be depicted is sparse. For this reason, NodeTrix (Henry *et al.*, 2007) was proposed, with the idea of depicting the dense portions of a graph with matrices, and the remaining portions in node-link form. Notice the analogy between elastic hierarchies (Zhao *et al.*, 2005) and NodeTrix (Henry *et al.*, 2007) : both allow two representations to be mixed in the depiction of the data, according to the density of data and the user's desires.

1.3.3 Visualization of Compound Graphs

There exists much previous work on the computation of hierarchical clusters of nodes within a graph (e.g., Auber *et al.* (2003); Gansner *et al.* (2005); Abello *et al.* (2006); Archambault *et al.* (2009)). The TreeMatrix technique that we present is concerned primarily with *how to depict* the resulting compound graph, and is independent of the method used to determine the clusters.

There is also previous literature on computing the layout of node-link diagrams of compound graphs (e.g., Sugiyama et Misue (1991); Sander (96); Bertault et Miller (1999)). More recent visualizations of compound graphs have proposed new variants of node-link diagrams (Holten, 2006; Pretorius et van Wijk, 2006; Balzer et Deussen, 2007; Greilich *et al.*, 2009) (see von Landesberger *et al.* (2010) for a survey of techniques for visualizing compound graphs, and Elmqvist et Fekete (2010) for a related survey of techniques for aggregation).

Lattix (Sangal *et al.*, 2005) is the earliest matrix-based technique we know of for visualizing compound graphs (Figure 1.4). Edges of the graph are weighted, and these weights are indica-

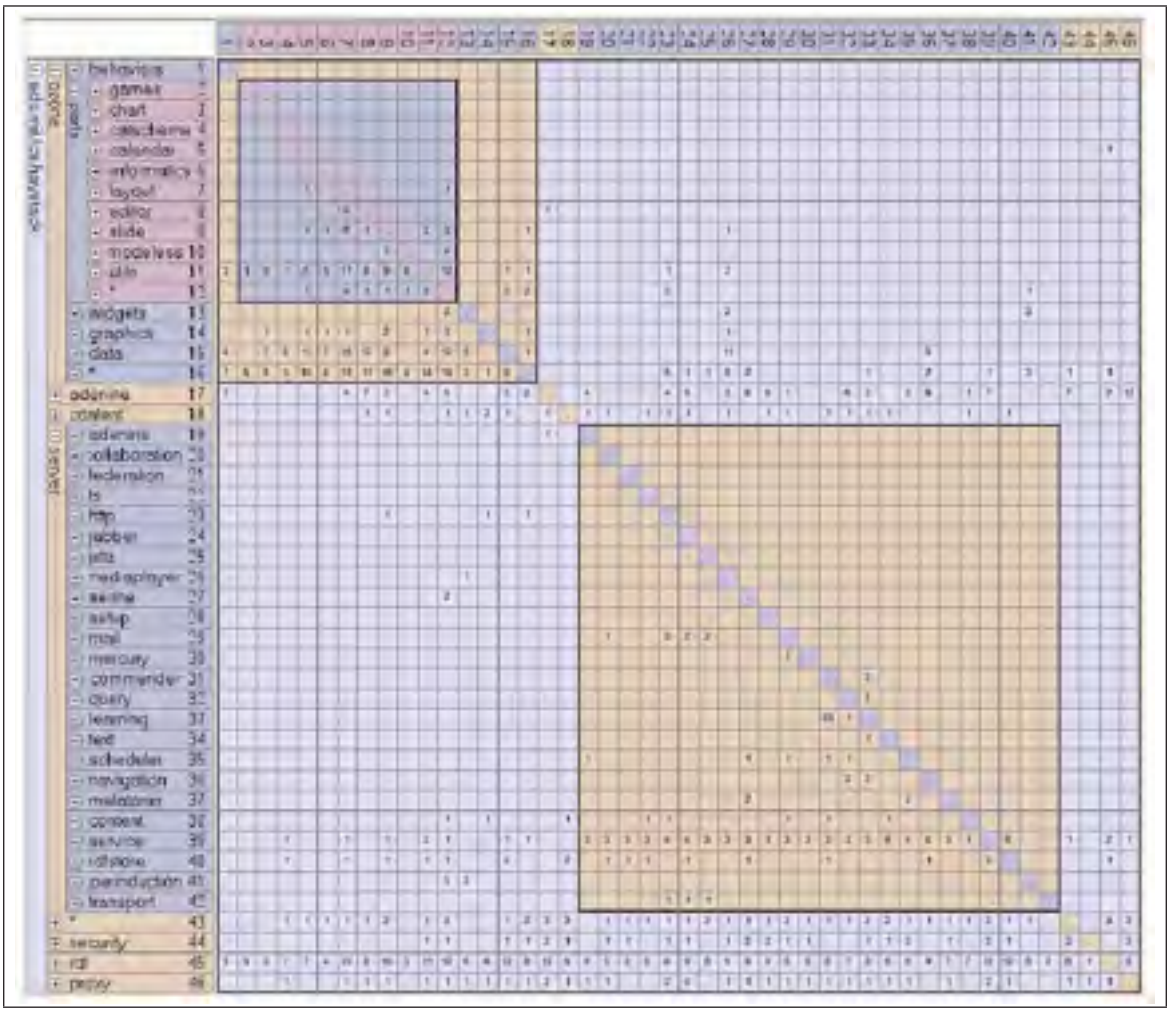

Figure 1.4 A screenshot of Lattix (Sangal *et al.*, 2005).

ted in the cells of the matrix. The tree structure is shown using an icicle diagram along the left of the matrix ; this icicle diagram can be used to expand or collapse portions of the compound graph. The tree structure is also shown, redundantly, with nested squares within the matrix.

Note that all of these previous approaches use only one representation for the graph structure of the compound graph. None of them are hybrid, in the sense of using a mixture of representations for either the tree structure, or for the graph structure. (Lattix does use two duplicated representations for its tree structure : an icicle diagram and nested squares, but these are not mixed in a way that allows a user to trade-off between them, as a hybrid like elastic hierarchies does.) Thus, these previous techniques will suffer from at least one of the disadvantages mentioned in sections 1.3.1-1.3.2. For example, most of the previously published techniques use node-link diagrams to show the graph structure of the compound graph, and thus will suffer from edge congestion if there are many edges. Lattix uses a single, large matrix for the entire compound graph, rather than a hybrid like MatLink or NodeTrix, and thus suffers from the disadvantages of matrices that paths are difficult to follow and the area required is $\Theta(N^2)$.

To summarize, our work is concerned not with clustering algorithms or layout algorithms, but hybrid depictions of the compound graph. A tantalizing possibility is to combine the approaches of elastic hierarchies and NodeTrix to show both the tree and graph structure of a compound graph in hybrid forms, allowing the user to use the most appropriate representations for each part of the data. As we will show, however, our final solution is not simply "elastic hierarchies + NodeTrix", as there are several subtle design issues to consider.

In the next section, we illustrate these issues with taxonomies of visualizations of trees and graphs. Contrary to the prior survey of techniques in von Landesberger *et al.* (2010), we will not consider techniques for time-varying structures in our taxonomy, but instead focus on constructing taxonomies of hybrid visualizations.

1.4 Taxonomy of Hybrid Visualizations of Trees and Graphs

Consider a dataset D and a function $A(D)$ that maps D to a visual representation. Consider further that some alternative visual representation may be used, given by function $B(D)$. To

create a hybrid mixture of the two representations, we take some subset $S \subset D$ and visualize it as $A(S)$, and visualize the remaining data $D \setminus S$ as $B(D \setminus S)$. Finally, some means is used to combine the two visualizations to yield a single hybrid $A(S) \oplus B(D \setminus S)$. The exploration of different hybrid visualizations for a given data type can thus be seen as choosing different functions for A and B, choosing an appropriate subset S that we may wish to represent differently from the rest of the data, and choosing one or more operators $\oplus$ for combining representations.

If D is a tree, then each of the functions A and B could be a node-link representation or a treemap representation (other representations of trees are possible, but for our purposes these two will provide a useful contrast). Many subsets S of the tree could be considered, however one subset that occurs naturally when dealing with tree data is a *subtree* of some node n. So, let S be the subtree $S(n)$ under node $n \in D$. We will refer to the remaining nodes $D \setminus S$ as the "surrounding nodes". Finally, in the expression $A(S(n)) \oplus B(D \setminus S(n))$, the operator $\oplus$ could either insert $A(S(n))$ under n's parent (call this $\oplus_{insert}$), or could display $A(S(n))$ some distance away from $B(D \setminus S(n))$ with a connective line (call this $\oplus_{connect}$). Choosing a hybrid is then equivalent to choosing between node-link and treemap for each of A and B, crossed with $\{\oplus_{insert}, \oplus_{connect}\}$, yielding 8 possible hybrid visualizations. Of these, two are eliminated as uninteresting because there is little difference between $\oplus_{insert}$ and $\oplus_{connect}$ when B is a node-link representation. The remaining 6 visualizations are shown in Figure 1.5.

An analogous taxonomy can be generated for hybrid visualizations of a graph D. In this case, the subset $S \subset D$ may be any subgraph, but would typically be a cluster of nodes that the user wishes to represent differently from the rest of the graph. We will refer to this subset S as a "local subgraph", since it may be thought of as a meaningful set of nodes that are somehow close to each other, perhaps close to a focal node. Next, the functions A and B can each be node-link or matrix representations, and the operators $\oplus_{insert}$ and $\oplus_{connect}$ are analogous to the previously defined ones. Again, 8 possible hybrids emerge, and again 2 can be eliminated because there is essentially no difference between the two operators if B is a node-link representation. Figure 1.6 shows the result.

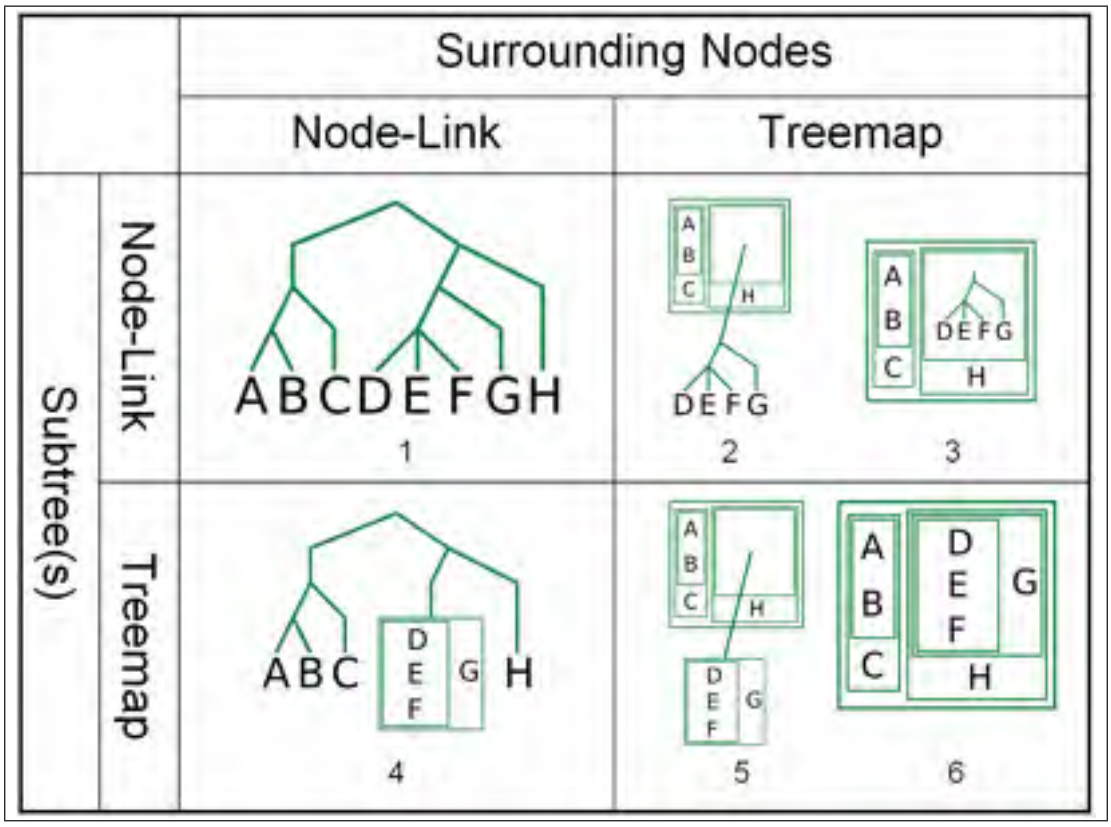

Figure 1.5 Hybrid visualizations of trees. 1 shows a classical layered node-link diagram, and 6 shows a "pure" treemap. 2, 3, 4, and 5 show various instances of hybrid Elastic Hierarchies (Zhao *et al.*, 2005).

Notice that the rows and columns of Figures 1.5 and 1.6 are analogous : the rows correspond to the choice for A, the columns to the choice of B, and in both figures the subfigures 2 and 5 are the result of using the $\oplus_{connect}$ operator rather than $\oplus_{insert}$. The $\oplus_{connect}$ operator draws the subset S as if it were "extracted" from the rest of the data, which could be useful for providing a kind of focus + context visualization.

Figure 1.7 pertains to compound graphs, having both a graph structure and tree structure. However, in this figure, each of the tree structures, and each of the graph structures, is shown using only *one* kind of visual representation, hence these are not hybrids in the sense of those in Figures 1.5 and 1.6. Notice also that, for tree structure, the categories "Treemap" and "Node-Link" of Figure 1.5 have been replaced with the more general categories "Nested Enclosure" and "Other", respectively, to allow for more possibilities in Figure 1.7.

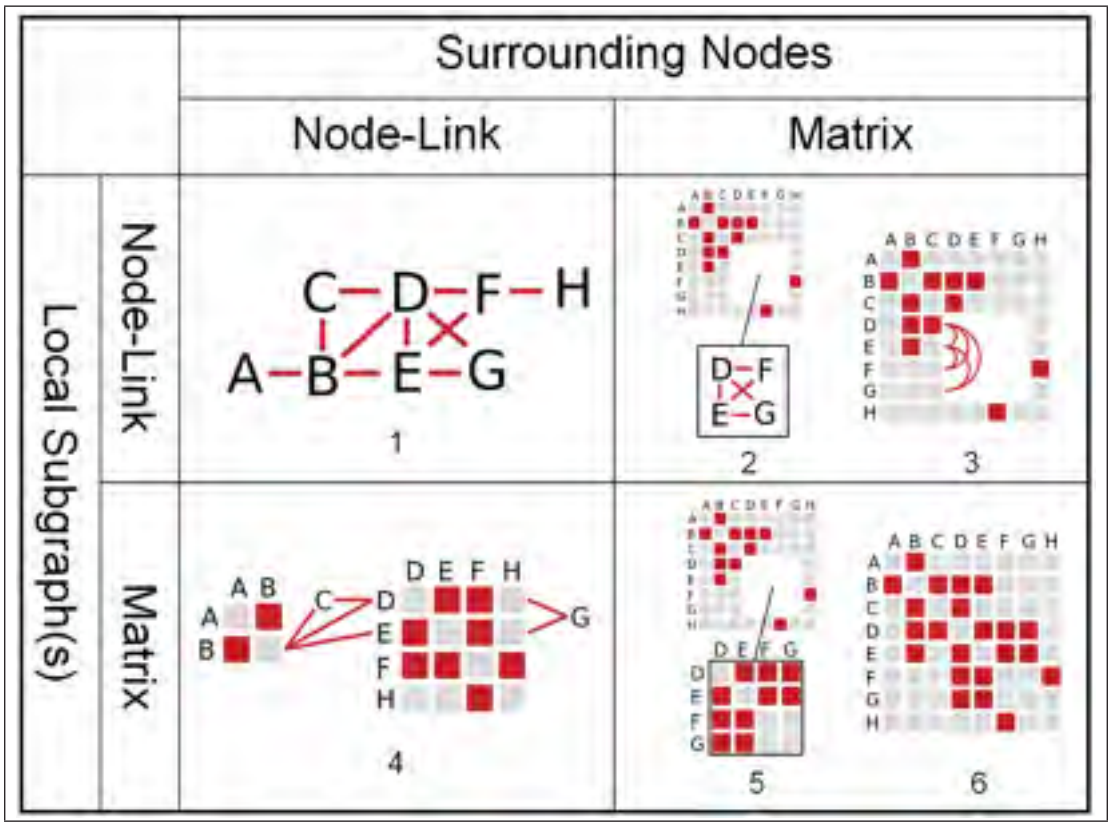

Figure 1.6 Hybrid visualizations of graphs. 1 shows a node-link diagram, and 6 shows a "pure" adjacency matrix. 4 shows NodeTrix (Henry *et al.*, 2007), and 2, 3, 5 show other possible hybrids.

To generate a taxonomy of *hybrid* visualizations of compound graphs, we again consider choices for the formula $A(S) \oplus B(D \setminus S)$. Let S again be a subtree $S(n)$ of some node n, which corresponds to a subset of both the tree structure and the graph structure. Notice that each of A and B may be any of the representations in Figure 1.7. Without even considering different choices of $\oplus$, we end up with a large space of possibilities, shown in Figure 1.8. Again, rows and columns correspond to choices of A and B, respectively.

Many possible hybrids exist in this space, and we have not tried to depict them. Notice simply that the non-hybrid forms of Figure 1.7 lie along the diagonal of this new taxonomy, whereas hybrids lie off-diagonal.

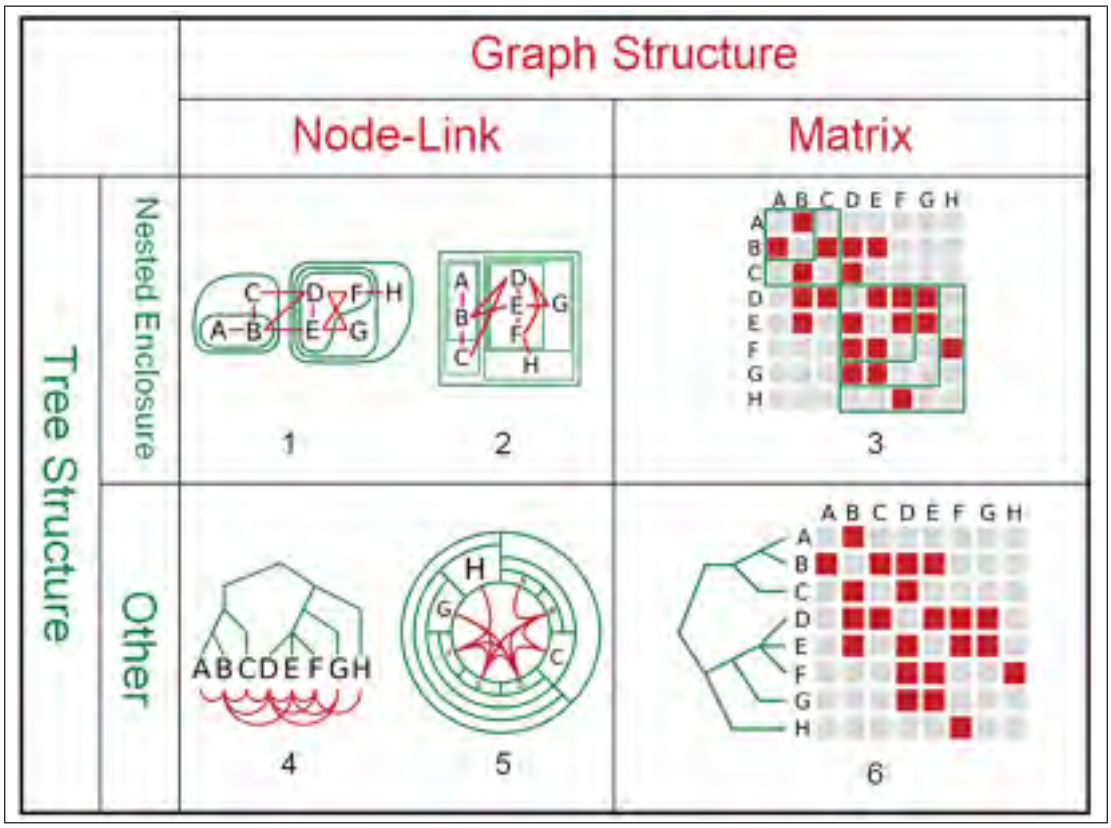

Figure 1.7 Non-hybrid visualizations of compound graphs showing both graph and tree structure. The lower row, labelled "Other", shows tree structures using node-link or icicle diagrams. 1 is comparable to Harel (1988); Sindre *et al.* (1993). 2 is similar to Fekete *et al.* (2003). 4 is similar to Pretorius et van Wijk (2006), and also similar to TimeArcTrees (Greilich *et al.*, 2009) if limited to only one moment in time, i.e., a node-link representation of a tree with arcs between the tree's leaf nodes to show graph edges. 5 is similar to the radial visualization shown in Holten (2006). 3 and 6 are approaches for showing tree structure within a matrix and are similar to Lattix (Sangal *et al.*, 2005).

To help understand Figures 1.5 through 1.8, we have always depicted the same underlying tree and/or graph structure with 8 (leaf) nodes, and have always used green for tree structure and red for graph structure. Figure captions also cite corresponding previous work, when applicable.

To decide which hybrids within Figure 1.8 would be useful, we can start with an approach in the spirit of NodeTrix : subtrees that correspond to dense subgraphs should be depicted using matrices, while the surrounding graph could be depicted in node-link form. This limits us to the lower left quarter of Figure 1.8. Next, to depict the tree structure outside the matrices,

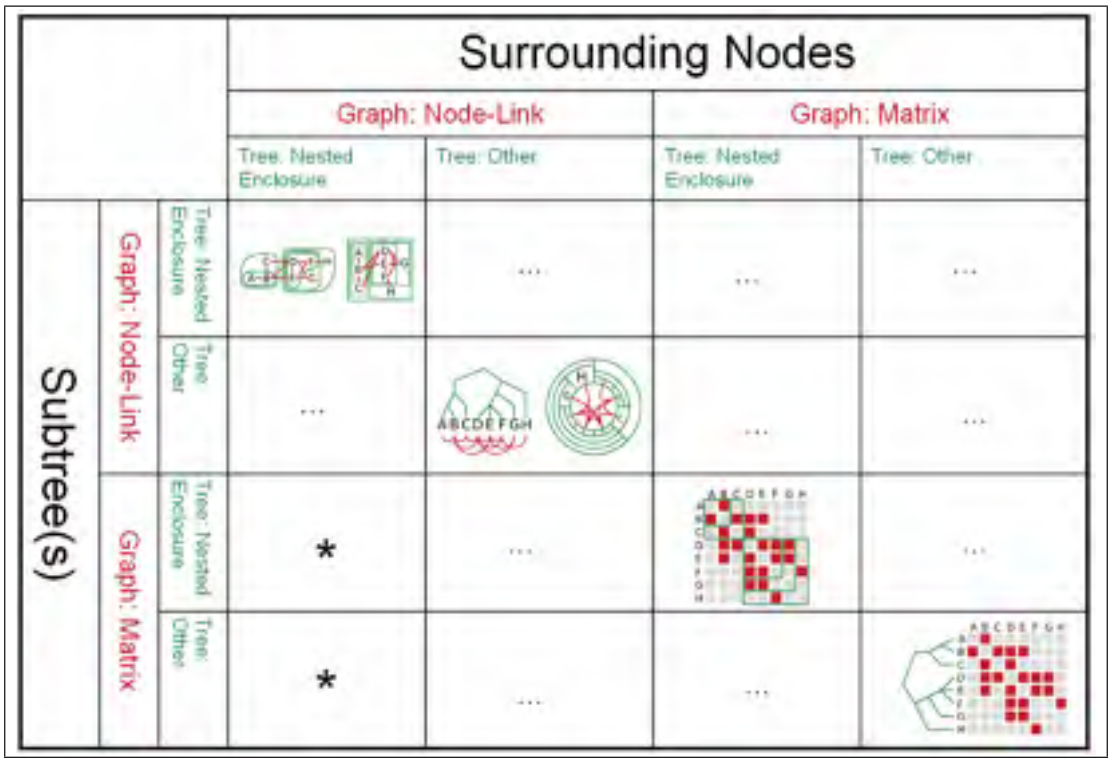

Figure 1.8 Hybrid visualizations of compound graphs. The TreeMatrix in Figure 1.9 (middle) corresponds to the two cells in the lower-left corner shown here containing stars ("*").

we choose to use a nested enclosure technique, since such representations are more space-efficient, as shown in McGuffin et Robert (2010). This further limits us to the left-most column of the taxonomy, i.e., the two starred ("*") cells in Figure 1.8. Notice that rather than having to choose between these two cells, we can happily combine the use of different tree visualizations. Finally, adding arcs along the edge of the matrix (in the spirit of MatLink) yields Figure 1.9 (top) for a single matrix, and Figure 1.9 (middle) for the entire compound graph. Notice that within each matrix in Figure 1.9 (middle), both the tree structure and the graph structure are shown twice, redundantly, using two techniques : nested squares and a node-link diagram of the tree, and matrix cells and arcs for the graph. The intention is that this double encoding will allow the user to attend to the most appropriate representation according to their current task and context. The combination of techniques in Figure 1.9 is thus the basis for the TreeMatrix visualization. We next describe our implementation.

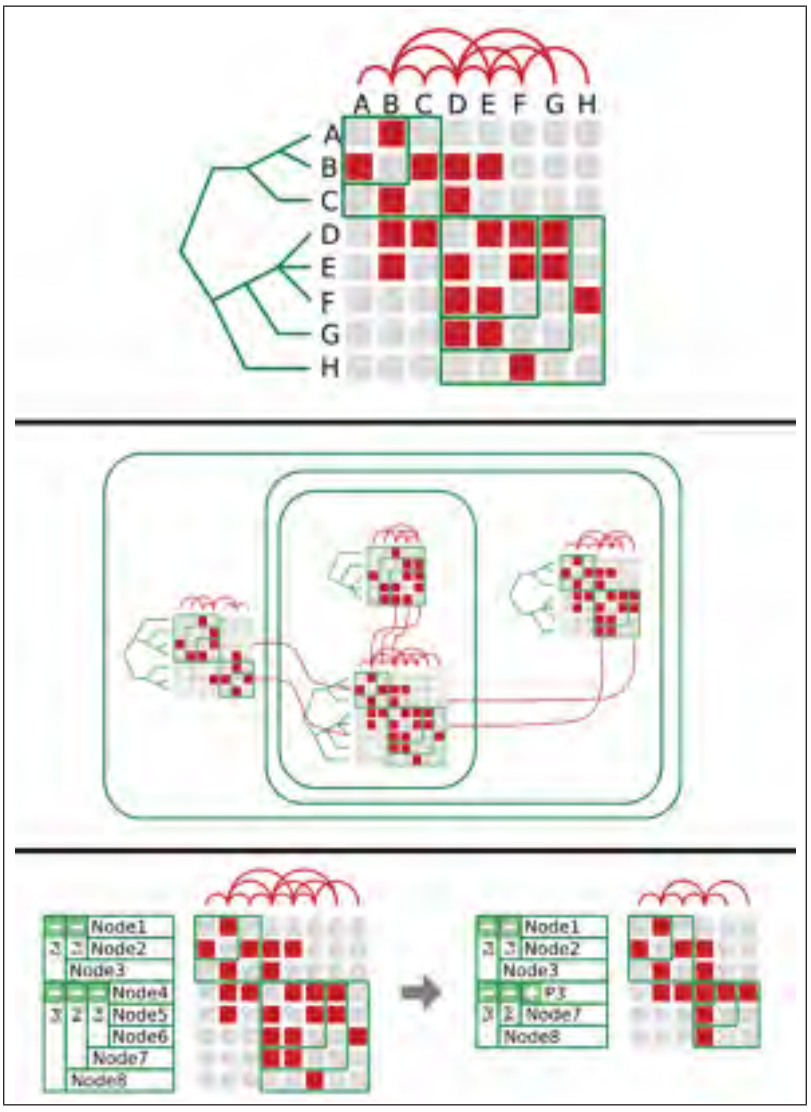

Figure 1.9 **Top :** each matrix in TreeMatrix depicts one cluster (possibly containing sub-clusters) of nodes, corresponding to a subtree of the compound graph's tree. The tree structure is shown in green, and is shown redundantly using both a node-link diagram to the left of the matrix, and nested squares within the matrix. Graph edges are shown in red, and shown redundantly using both filled-in matrix cells and arcs along the top of the matrix. **Middle :** An entire compound graph is shown using multiple matrices, with graph edges between matrices shown in red, and the tree structure outside matrices shown with green rounded rectangles. **Bottom :** Our prototype implements a slight modification to the design, replacing the node-link tree diagrams with icicle diagrams along the left edge of each matrix (see also Figure 1.2). The matrix here is shown before and after collapsing the parent node P3.

1.5 Prototype

Our prototype (Figures 1.1, 1.2) was implemented in Java using the Prefuse toolkit (Heer *et al.*, 2005). The compound graph visualized is a directed graph (hence the adjacency matrices are not symmetrical) with weighted edges. Notice that along the left edge of each matrix, we display an icicle tree diagram (Figure 1.9(bottom)) rather than a node-link diagram (Figure 1.9(top)), as the icicle diagram allows labels to be displayed more easily and interferes much less with inter-matrix edges. Small $+/-$ buttons within the icicle diagrams allow subtrees of each matrix to be expanded or collapsed. This kind of icicle diagram is also shown along the left edge of the matrix in Lattix (Figure 1.4).

In each matrix, the weight of each edge is shown in three ways : as a numeric value in the appropriate cell, by the background color of that cell (using a continuous grayscale), and by the color of the corresponding 180-degree arc. Between matrices, the colors of the edges again reflect their weight. The direction of edges is shown with small arrow heads. Edges drawn between matrices are drawn as cubic bezier curves. The start and end point of each of these bezier curves may be along any of the four sides (top, bottom, left or right edge) of the relevant matrix node ; the side used is chosen to minimize curve length. When drawing aggregated edges, start and end points are made to coincide to bundle the edges together, reducing clutter. The user may interactively control such aggregation of edges, by collapsing matrices (e.g., the edges from the “Sys1”, “Data”, “Util”, and “org” nodes in Figure 1.1 are aggregated because those nodes have been collapsed), allowing the user to reduce clutter and focus only on the nodes of interest. To further reduce clutter from edges, a mode can be activated such that only the node under the mouse cursor has its 180-degree arcs drawn, rather than all such arcs.

When the mouse hovers over a node, that node is highlighted. The user may select individual nodes by clicking, or a contiguous set of nodes in the matrix by click-dragging and using lasso gestures.

Once selected, nodes can be manipulated using actions within the radial menu (Callahan *et al.*, 1988; Kurtenbach et Buxton, 1993) shown in Figure 1.1. Placing these actions in a radial menu

rather than a toolbar has several advantages. Radial popup menus require no screen space when not in use. Also, they eliminate the need to travel back and forth between a work area and a toolbar. Finally, they can be invoked with rapid directional drags.

In our prototype, the user may also interactively collapse or expand subtrees, as well as change the location of nodes or highlight them, all to change the graphical presentation of the compound graph. Every time a new matrix is instantiated, the rows and columns can be automatically re-ordered using the barycentric algorithm (Sugiyama *et al.*, 1981a; Mäkinen et Siirtola, 2000), to make highly-linked groups of nodes more evident within the matrix (see section 4.2 of Henry (2008) for an overview of matrix ordering algorithms). To ensure that the ordering does not violate the hierarchical grouping of nodes within the matrix, the barycentric ordering algorithm is only applied to the leaf nodes of the lowest levels of the tree of clusters. The user may manually reorder the rows and columns. All these operations allow the user to manage available screen space and represent the data in whatever way best for their task, however these operations only change the presentation of the data and do not change the *structure* of the compound graph.

In addition, the user may interactively change the *structure* of the compound graph, by selecting any subtree and moving it under a new parent node, by selecting a set of nodes and moving them under a new subtree, or by splitting or merging nodes (Figures 1.10, 1.11). This allows the user to change the way the compound graph is clustered. Such operations are useful in software engineering when performing software design discovery.

Finally, a text field in the upper right corner of the main window (Figure 1.1) allows the user to search for nodes by name. Matching nodes are highlighted, unless they are hidden because a parent or ancestor has been collapsed, in which case the nearest visible ancestor node is highlighted. The right margin of the main window (Figure 1.1) also contains an overview of the entire compound graph, which is useful for navigating a zoomed-in view in the main window (in other words, the user has a a focus + context interface).

Figure 1.10 Moving nodes and changing their representations. **A** : the user may select a subset of nodes (in this case, the last four nodes), and move them into a destination rounded rectangle by selecting "Move Here" in the radial menu (in this case, creating a 4×4 matrix). **B** : the user selects one subtree within the matrix (by moving the cursor over the corresponding square) and selects "To Matrix" in the radial menu, causing the subtree to be extracted as a new, smaller matrix. **C** : dragging an existing matrix or rounded rectangle inside another rounded rectangle causes the dragged cluster to be re-parented under the dropped location. **D** : dragging an existing matrix or rounded rectangle toward the border of its parent rounded rectangle has one of two effects : if dragged towards its parent, the parent increases in size and continues to contain the child ; if dragged towards another parent, the child is allowed to leave the parent and is re-parented under that other rounded rectangle. **E** : the user can also transform a cluster of nodes back into a matrix.

1.5.1 Software Design Discovery

While any compound graph can be visualized using our prototype, we were interested in evaluating our approach in the software design domain. The data shown in Figures 1.1, 1.2, 1.10, 1.11 is the source code structure of the ApexText text editor program (http://sf.net/projects/apextext/).

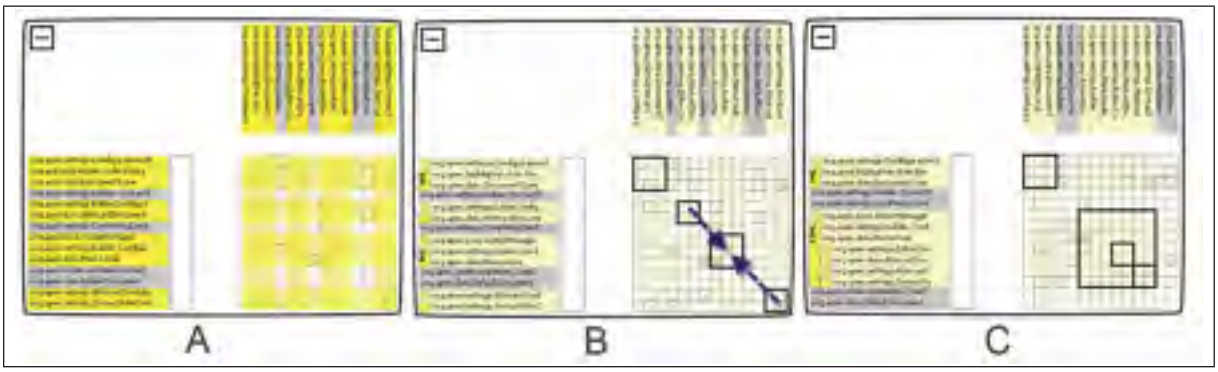

Figure 1.11 Creating and editing of subtrees within a matrix. **A** : initially, this matrix contains no hierarchy. The user selects 4 different subsets of nodes and creates a subtree for each one, shown in B. **B** : tabs are visible along the left edge of the matrix. These form the icicle diagram showing the hierarchy, and indicate there are now 4 subgroups. To edit the hierarchy, the user may drag groups within their parent or into each other, resulting in C. **C** : the icicle diagram along the left edge of the matrix indicates the new hierarchical organization of groups within the matrix.

Each node is a Java class, and each directed edge is a *coupling* between the software elements, computed with a source code analysis tool. Edge weights are computed from the number of couplings between elements.

Initially, the hierarchical clustering of nodes can be automatically deduced from the organization of packages (subdirectories) of source code classes. The user can then browse the compound graph to see if the couplings (edges) either respect or violate the package architecture of the software. As described in the previous section, the user can also dynamically (re-)define the hierarchical clustering of nodes at runtime, which is useful for software design discovery where the user interprets the couplings and chooses the most logical clustering of nodes, and also useful if the user sees that the package architecture is violated by many couplings and should be reorganized.

We note that in software design discovery, the typical workflow is to progress one layer of abstraction at a time, and examine the links *between* clusters (rather than those within multiple clusters) to determine how to organize the next layer. This workflow is supported through our prototype's ability to collapse any subtree, hiding the details inside that subtree and aggregating (and bundling) edges emanating from that subtree, thereby reducing clutter and allowing the

user to focus on links between that cluster and others. The ability to visualize any subtree as a matrix also reduces clutter by eliminating edge crossings within that subtree.

1.6 Comparison with Lattix

Lattix (Sangal *et al.*, 2005) is a commercial software package for visualizing a compound graph representing the couplings between source code modules and their hierarchical clustering (Figure 1.4). Lattix is useful for software design recovery, and allows the user to dynamically change the hierarchical clustering of nodes, as does our TreeMatrix prototype. However, as mentioned earlier, Lattix visualizes the compound graph as one large adjacency matrix, called the "Design Structure Matrix" (DSM).

Several DSM analysis and visualization tools are listed at http://www.dsmweb.org. To our knowledge, Lattix is the only one of these that supports software design recovery tasks (such as manually changing the hierarchical clustering of modules), and is also the most widely-used software analysis tool that uses a matrix-based visualization. Thus, it is a reasonable candidate for comparison with TreeMatrix. In addition, to our knowledge, there are no DSM tools that allow the adjacency matrix to be split apart and visualized as multiple small matrices, as does TreeMatrix. The ability to split the matrix into multiple matrices in TreeMatrix means that users may position two or more matrices of interest close to each other to see the edges connecting them, whereas in Lattix it can be very inconvenient or impossible to see the edges connecting multiple groups of nodes, hence users of Lattix may sometimes need to do much scrolling, resulting in loss of context. The increased flexibility afforded by TreeMatrix's hybrid visualization is the primary difference between our prototype and Lattix that we are interested in evaluating.

There are several secondary differences between the TreeMatrix prototype and Lattix. For example, TreeMatrix has a popup radial menu, whereas Lattix uses traditional pull-down menus and toolbars to access functionality. Also, both software tools display the numerical weight of each edge (i.e., the number of couplings) within the appropriate matrix cell, however TreeMatrix *also* shows this weight by coloring the cell, causing interesting weight values to "pop

out" visually. This means that groups of interesting edges (and their associated nodes) can be perceived faster in the TreeMatrix prototype for subsequent selection and manipulation. Finally, TreeMatrix allows for both pan and zoom to be done continuously, whereas Lattix only allows continous panning (or scrolling).

1.7 Qualitative User Study

We compared our TreeMatrix prototype with Lattix version 6.2.6 in a qualitative user study involving tasks related to software design discovery. Ten participants (P1-P10) who were students in a master's-level course on software design received three hours of instruction on Design Structure Matrix (DSM) theory and layered architectural style for software design. The students were separated into two groups (A and B) of five students each. Two professional software engineering researchers (P11 and P12) also participated in the study bringing the total number of participants to 12. The study involved three design discovery tasks, performed on the source code of two open-source software projects (orion-ssh2 and sdedit).

1.7.1 Choice of Source Code Projects

We selected the data sets among the open-source projects available on SourceForge. Specifically, we sought two projects that were (1) in development for more than a year, (2) developed in Java, (3) used in different application domains, and (4) contain between 100 and 200 classes. The first project is OrionSSH2 (build 213), a library for the SSH2 protocol, and is available at http://sf.net/projects/orion-ssh2/ . The second project is an UML editor program, Quick Sequence Diagram Editor 3.0.5, available at http://sf.net/projects/sdedit/ .

1.7.2 Tasks

We chose three tasks to be done using each tool, which were performed by participants in each group during sessions that lasted 3.5 hours. In each session, the participants received 30 minutes of training (5 minutes per task, per tool) and had 1.5 hours per tool to perform the tasks and fill out questionnaires (approximately 30 minutes per task).

In the first task, the participants had to categorize the relationships :

a. Select all the nodes in the matrix and cluster them ;

b. Find a node with very little incoming links (type 1) ;

c. Find a node with very little outgoing links (type 2) ;

d. Find a cycle or type 3 category (i.e., nodes with a high density of edges along the matrix's diagonal).

The second task aimed at restructuring the design in a layered fashion. In a software architecture, the direction of the links in a design should be from a top layer to a bottom layer (Garlan et Shaw, 1993). The participants had to :

a. Check which of the following scenarios apply :
   a) If the node of type 1 identified previously uses other elements, it is an upper layer. Move it above the first used element.
   b) If the node of type 2 identified previously is used by other elements, it is a lower layer. Move it below the last element which uses it.
   c) Group the nodes inside the cycle (type 3).

b. Find a new cycle or type 3 and group these nodes ;

c. Find a node with a strong link with the group created in the previous step and move that element in the group.

In the third task, they had to explore high level and low level links between nodes and restructure the design. They were asked to :

a. Identify the three top layers with the highest number of internal links.

b. Indicate the links between the three top layers and group the two more related layers as "TL".

c. Find the low level nodes which are responsible for the previous relationship. Group and place them under “TL”.

d. Find the nodes which are linked to the “TL” group and move them into it.

1.8 Results

In this study, we collected user feedback, user ratings (Table 1.1), and task completion times (Table 1.2).

Tableau 1.1 Average ratings by participants of how each tool performed for components of the tasks, on a scale of 1 (Poor) to 10 (Excellent), along with the p-values of the Wilcoxon signed rank test.

	Task Component	Lattix	TreeMatrix
1	Perception of relationships within matrices (Tasks 1, 2)	8.50	5.75 p = 0.0051* (n=12)
2	Ability to reorganize matrices (Task 2)	8.50	6.17 p = 0.0054* (n=12)
3	Identification of high-level links (links between clusters) (Task 3)	6.75	8.17 p = 0.0381* (n=12)
4	Identification of low-level links (links between classes) (Task 3)	7.42	7.42 p = 0.9682 (n=12)
5	Ability to reorganize entire clusters at a high level (Task 3)	7.58	7.58 p = 0.9683 (n=12)
6	Highlighting of nodes and edges (Tasks 1-3)	6.80	7.80 p = 0.1649 (n=10)

Tableau 1.2 Average timings by participants of how each tool performed with respect to tasks (in minutes), along with the p-values of the Wilcoxon signed rank test.

	Task	Lattix	TreeMatrix
1	Find clusters in a matrix	14.25	11.63 p = 0.4349 (n=8)
2	Reorganize nodes inside a matrix	25.11	18.22 p = 0.0116* (n=9)
3	Restructure the design	20.92	19.70 p = 0.1381 (n=10)

Users rated the usefulness of the hybrid of matrices in TreeMatrix at 2.2, on average, on a scale of 0 to 3 (unused, not very useful, useful, very useful). Some participants explained that our visual approach is advantageous in that the ability to split matrices helps to have a better view of the design elements and their relationships (P1-P6, P9). For example, participants mentioned that having multiple matrices facilitates the visualization of links between software layers (P6) and the identification of low level elements that are responsible for higher level couplings (P1).

The 180-degree arcs in TreeMatrix were found useful by 75% of the participants (P1-P5, P8, P10, P11, P12) and they noted that "Arcs help to find relationships between elements and evaluate the link strength" (P1, P2, P8, P10 all made similar statements) and that "it is a huge plus" (P1, P4).

Some users also expressed that they preferred the arrow heads showing edge direction in TreeMatrix, rather than having to deduce the direction of edges based on their location in Lattix's matrix. For instance, P4 and P9 commented that arrows are easier to follow than reading a matrix cell. Recall that, in TreeMatrix, the weight of an edge in a matrix cell is shown both numerically and with a color, whereas in Lattix it is only shown numerically. One user (P12) suggested that the TreeMatrix prototype should also show numerical weight values beside the 180-degree arcs and bezier curves that are outside each matrix, to reveal detailed information. Two users (P6, P7) asserted that the tooltips, supported in both tools, helped them to read the direction of an edge. All participants were able to learn how to use the radial menu although they had never used one before. Participants also appreciated the ability to collapse subtrees and aggregate edges to reduce the visual complexity.

1.8.1 Tasks Ratings and completion times

The task completion results show that tasks were, on average, faster using TreeMatrix. However, there were no statistically significant differences for tasks 1 and 3. The first two task components focus on a matrice perspective and users generally preferred Lattix for these (Table 1.1, Task Components 1 and 2). Participants explained that they rated TreeMatrix less positively because its non-traditional user interface made them feel less confident. Reasons given included

that the prototype uses a novel radial menu, in contrast to the traditional menu bars of Lattix ; and TreeMatrix also lacks keyboard shortcuts, undo, and traditional scrollbars for scrolling, which are features of Lattix. In addition, some participants (P1, P7, P10) pointed out that Lattix makes it easy to see the diagonal of the matrix at all times, which can help to find cycles. We find interesting that the more classical approach of Lattix in tasks 1 and 2 was not faster, despite the unfamiliar aspects of TreeMatrix that the users had to learn. The second task was significatively faster using TreeMatrix and we believe this is due to the highlighting and the 180-degree arcs that were appreciated.

The ratings for the 3rd task component show that TreeMatrix was preferred for analyzing links at a higher level of abstraction. This is backed up by some participants commenting that TreeMatrix is better for qualitative analysis (P2, P10). The TreeMatrix prototype was also appreciated with respect to highlighting of edges and nodes (the 6th component in Table 1.1). Participants explained that “the colored edges and the inter-matrix links really help” (P1, P6) and commented that the TreeMatrix prototype is more visual, helping to perform the tasks (P3, P7, P9). The task completion results show that TreeMatrix was not significantely faster for the third task. The comments suggest that some users have spent more time in visually exploring the data set and playing with the interface.

We believe that certain modifications could be easily made to the TreeMatrix prototype to increase its usefulness to users : implementing undo, keyboard shortcuts, optional traditional scrollbars and pull-down menus, making the diagonal of each matrix always easier to see, and displaying the edge weights of bezier curves and arcs with numerical labels (possibly displayed in a tooltip).

1.8.2 Threats to validity

We first consider the internal validity (i.e., the confidence level of the results) of the user study. We tried to prevent the effects of confounding variables, between a tool (independent variable) and a dependent variable (e.g., user ratings, time). Participants received the same training for both approaches. None of the ten students had any prior experience with TreeMatrix or Lattix,

while the two researchers had experience using Lattix. The participants had no prior exposure of the open-source projects. To counterbalance the conditions, the participants of each group were randomly chosen, with group A performing the three tasks using Lattix on orion-ssh2 and then using TreeMatrix on sdedit. Group B performed the tasks on the same data sets, but started with TreeMatrix and then with Lattix. There were no significant differences among the expert users, compared to the other users.

We statistically verified the differences in the dependent variables. We verified that we could perform the tasks for all projects with a similar effort and time. Given the exploratory nature of the tasks and the real-world complexity in the data, we focused on collecting the impressions of the participants after performing useful exploratory tasks and did not evaluate the error rates. However, the user feedback that we collected suggest that there were no significant differences among the tools. The participants had to write down their timings at each substep and we excluded any uncertain data.

We now turn to the external validity (the generalizability of the study). Real and complete projects, from different domains, were evaluated by the participants in all their complexity, which should ensure that the approach has a practical value for other projects. Since a small number of partipants were involved, it should be repeated with more participants. The prototype, demonstrated in the supplementary video [1], has been tested with software designs of a few thousands of nodes and links on a Sager NP8150 laptop. We valided one hybrid approach, but further research is needed to evaluate the other possible hybrids that can be derived from our taxonomy againsts a larger number of tools and domains.

1.9 Conclusions and Future Directions

We have presented new taxonomies of hybrid visualizations of trees, graphs, and compound graphs, and shown how these taxonomies lead to a novel hybrid visualization of compound graphs called TreeMatrix. TreeMatrix combines advantages of NodeTrix and MatLink (which, in their original form, are not designed for compound graphs) with the matrix visualization of

1. The video is available online at http://ref.rufiange.com/cgf2012.

Lattix. Our hybrid visualization allows users to represent selected clusters of the compound graph as matrices, with each matrix providing a visual summary of its contents, free of edge crossings, and containing a double encoding of both the tree and graph structure within the matrix.

We have also presented a prototype implementation of TreeMatrix, and demonstrated how it can be useful for software design discovery tasks, such as visualizing the structure of source code. Our prototype displays both low-level and high-level abstractions (within and above the level of matrices, respectively) in one integrated view, and allows the user to dynamically edit tree clusters (inside a matrix or at an upper level).

We reported the results of a comparative evaluation of our prototype with Lattix, a commercial-grade, matrix-based software tool. Lattix was generally preferred by participants for lower level tasks, such as reorganizing a single matrix.

Several users found our prototype's ability to split matrices into multiple submatrices advantageous, and found it either useful or preferable that our prototype displays edges as arcs or curves with arrowheads. User ratings indicate that they preferred our prototype in particular for interpreting high-level links between clusters. This confirms the rationale behind using a hybrid visualization : TreeMatrix has the flexibility of enabling low-level details to be shown within matrices (eliminating edge crossings) while allowing high-level edges to be shown in node-link form (which are easier to interpret, if they are not too dense or numerous).

Based on some of the comments we collected, one possible future direction would be to allow the user to switch between novice or advanced modes to access a different set of features. There are also improvements that could be made to the computed layout and bundling of edges, to make the TreeMatrix visualization even less cluttered. Future user studies could also examine in more detail the kinds of tasks (Lee *et al.*, 2006) that benefit from hybrid visualizations, e.g., comparing hybrid and non-hybrid visualizations in a follow-up to Archambault *et al.* (2010).

Another future direction would be to implement an interactive prototype flexible enough to allow the user to explore all possible hybrid combinations in Figure 1.8, with automatic ge-

neration of appropriate layouts and even smoothly animated transitions during changes to the visualization. Although many of the hybrids generated this way might not be very useful, there is the possibility that new useful forms could be discovered this way.

CHAPITRE 2

ARTICLE 2. DIFFANI : VISUALIZING DYNAMIC GRAPHS WITH A HYBRID OF DIFFERENCE MAPS AND ANIMATION

Sébastien Rufiange[1] and Michael J. McGuffin[1]

[1]Department of Software and IT Engineering, École de technologie supérieure,

1100, rue Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

sebastien@rufiange.com, michael.mcguffin@etsmtl.ca

Article soumis à la conférence “InfoVis 2013” (actes publiés dans la revue “IEEE Transactions on Visualization and Computer Graphics”) et accepté le 11 juillet 2013.

2.1 Abstract

Visualization of dynamically changing networks (graphs) is a significant challenge for researchers. Previous work has experimentally compared animation, small multiples, and other techniques, and found trade-offs between these. One potential way to avoid such trade-offs is to combine previous techniques in a hybrid visualization. We present two taxonomies of visualizations of dynamic graphs : one of non-hybrid techniques, and one of hybrid techniques. We also describe a prototype, called DiffAni, that allows a graph to be visualized as a sequence of three kinds of tiles : diff tiles that show difference maps over some time interval, animation tiles that show the evolution of the graph over some time interval, and small multiple tiles that show the graph state at an individual time slice. This sequence of tiles is ordered by time and covers all time slices in the data. An experimental evaluation of DiffAni shows that our hybrid approach has advantages over non-hybrid techniques in certain cases.

Keywords : information visualization, network, evolution, taxonomy, hybrid visualization, dynamic graphs, animation, difference map.

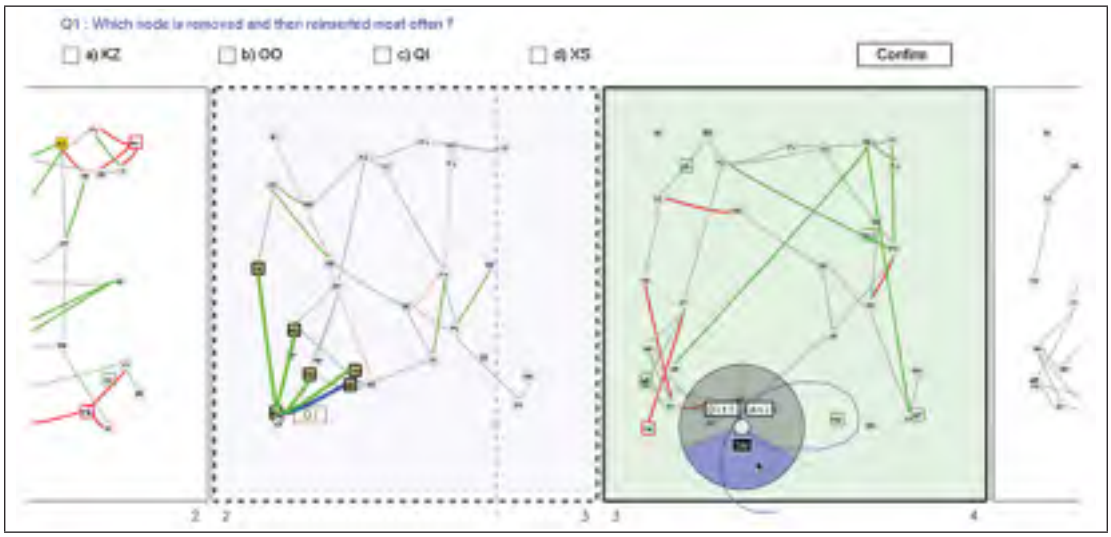

Figure 2.1 Given a dynamic graph defined over a series of time slices, our DiffAni hybrid visualization displays it as a sequence of consecutive tiles. Each tile may show one or more time slices, and there are three kinds of tiles : diff tiles (indicated with a solid border and two time slice numbers below them, e.g., the left-most tile), animation tiles (indicated with a dashed border and two time slice numbers below them, e.g., the tile covering time slices 2 to 3), and small multiple tiles (indicated with a solid border and a single time slice number below them, e.g., the right-most tile). Nodes and edges are colored in red if they are being removed, or green if they are being added, over time slices. The radial menu above is being used to convert a diff tile into two small multiple ("SM") tiles.

2.2 Introduction

Within the field of graph visualization (von Landesberger *et al.*, 2010), a significant frontier for research is dealing with time-dependent graph data. Many real-world networks change over time, including social networks, communication networks, migration of people between cities or countries, international trade networks, and network models of the relationships between source code modules. The visualization of such dynamic graphs is challenging for at least two reasons. First, in computing the layout of a graph for each moment in time (each time slice), there is a trade-off between optimizing the layout quality for that particular time slice, versus reducing the movement of nodes across time slices to improve the users' *mental map*. So far, empirical evaluations (e.g., Purchase et Samra (2008); Purchase *et al.* (2006); Archambault

et al. (2011a)) have not yielded simple, clear conclusions about the effect or optimal level of mental map preservation.

A second challenge is that there are several ways the time slices of a graph can be visually presented, including small multiples, animation, and 3D representations. Here again, there are trade-offs, for example : when compared to animation, small multiples require more space (or they require the user to sacrifice spatial resolution to fit all of the small time slices on a single screen), however small multiples have the advantage that the user can compare different time slices with fast eye movements rather than waiting for an animation to complete or replay. In addition, as with the first challenge mentioned, previous empirical comparisons (Archambault *et al.*, 2011a; Farrugia et Quigley, 2011; Zaman *et al.*, 2011) of different representation methods have yielded mixed results. It seems likely that the relative advantage of each visual representation depends on several factors, including the size and density of the graph, the degree of mental map preservation, and the task being performed by the user. For example, if the user is interested in the total number of nodes in a graph as it changes over time, then tracking individual node movements is not necessary, and a small multiples representation may be best. However, if the user must track the movements of a particular node, without the benefit of highlighting, then smooth animation may prove superior.

Previous studies seem to support this possibility, with some (Archambault *et al.*, 2011a; Farrugia et Quigley, 2011) finding that static representations are superior, and another (Zaman *et al.*, 2011) finding that animation is better. A reasonable user interface, therefore, might allow users to view a dynamic graph either as small multiples or as an animation, or even show both in side-by-side coordinated views. This would present other problems, however. Different subsets of the dynamic graph data might best be visualized with different representations. Showing only one representation at a time in a single view could require the user to frequently switch between representations, as well as cause disorientation during switching. On the other hand, using multiple coordinated views would consume more screen space.

We therefore propose mixing representations together in a novel hybrid visualization of dynamic graphs. We have implemented this idea in a prototype called DiffAni (pronounced to

rhyme with Tiffany). DiffAni displays the network as a horizontal sequence of *tiles*, where each tile is either (1) a static small multiple showing a single time slice, (2) a static difference map that is colored to show differences between two time slices (we call these diff tiles), or (3) an animation clip that smoothly interpolates between two time slices. These three kinds of tiles can be intermixed, but always show time slices in their chronological order, from left to right, with the left-most tile corresponding to the beginning of the dataset, and the right-most corresponding to the end. Such a hybrid visualization gives the user the flexibility to change the representation over any time interval, and has the potential to display each temporal portion of the network with the best representation for it, without consuming the additional screen space that would be required with a multiple coordinated views approach.

The rest of this paper presents our contributions, which are (1) a taxonomy of non-hybrid strategies for visualizing dynamic graphs ; (2) a taxonomy of hybrid visualizations involving focal and context regions ; (3) our prototype DiffAni that allows users to select any consecutive set of time slices and change their representation, and also allows users to scroll across multiple "diff" frames and navigate within multiple animation frames all with a single, unbroken mouse drag gesture ; and (4) the results of a controlled experiment that show that DiffAni can sometimes yield performance superior to that with a non-hybrid visualization.

2.3 Related Work

2.3.1 Visualization of Dynamic Graphs

Visualization of time-varying data, in general, is of growing concern, as evidenced by a recent taxonomy (Cottam *et al.*, 2012). In the specific case of graphs, a dynamic network (Burch *et al.*, 2011; Sallaberry *et al.*, 2013) can evolve in different ways over time. Most previous approaches for visualizing dynamic graphs can be classified according to our own taxonomy in Figure 2.2, which shows each node with a changing color representing a numerical attribute. Other kinds of network changes, such as topological changes, are of course also possible, and will be the focus of section 2.4 and the later sections of this paper. Small multiples (Figure 2.2.1) show snapshots side-by-side, and can be thought of as nesting 2D layouts of the

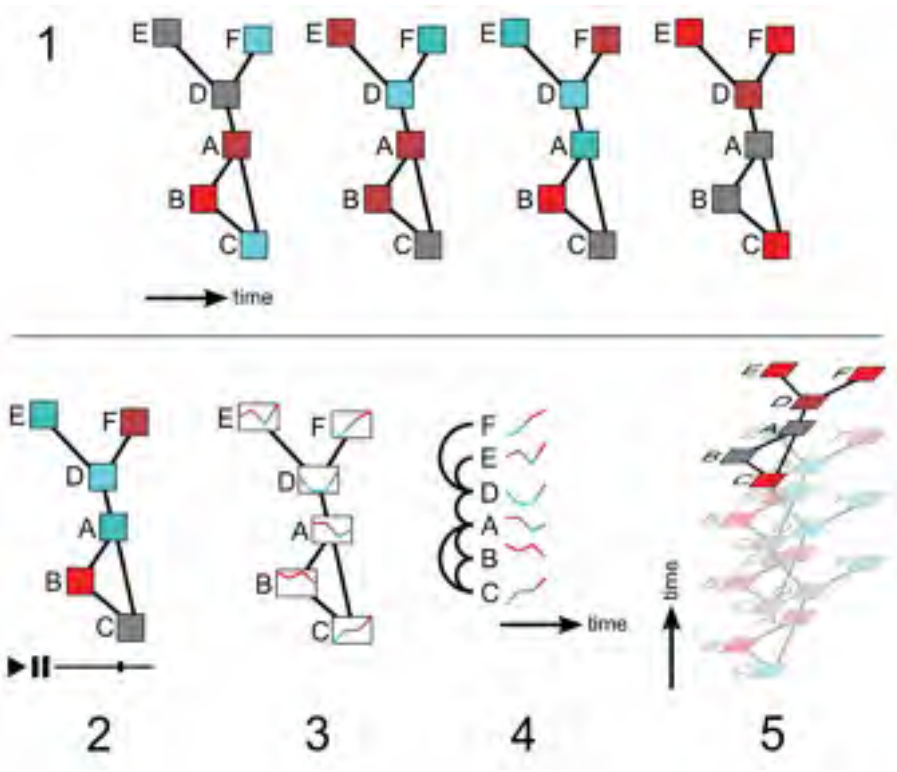

Figure 2.2 A taxonomy of strategies for visualizing dynamic graphs. Colors indicate a changing numerical attribute associated with each node. (For simplicity, the above figure shows changes in node attributes, however most of the above approaches could be adapted to instead show changes in topology, which are the kinds of changes discussed in the rest of this paper.) **1** : small multiples. **2** : animation. **3** : sparklines or other glyphs embedded within the graph's layout. **4** : the time axis is perpendicular to an arc diagram. **5** : a 3D visualization, with time along the 3rd spatial axis.

graph within an external time axis. The opposite approach is to nest the time axis within the elements of a 2D layout (Figure 2.2.3). The graph's time axis can also be mapped to *user* time, resulting in an animation (Figure 2.2.2), or to a 3rd spatial axis, resulting in a 3D visualization (Figure 2.2.5). Finally, if the graph is linearized along a single axis, as with an arc diagram (Bertin, 1967; Wattenberg, 2002), the 2nd spatial axis can be used for time (Figure 2.2.4). Variants of these approaches can also be created using adjacency matrices rather than node-link diagrams.

Small multiples and animation (Figures 2.2.1 and 2.2.2) have been the most common strategies in previous work (Purchase et Samra, 2008; Archambault *et al.*, 2011a; Farrugia et Quigley,

2011; Zaman *et al.*, 2011). Examples of nesting the time axis inside the network layout (Figure 2.2.3) include (Saraiya *et al.*, 2005b) (which uses a node-link representation for the graph) and (Yi *et al.*, 2010; Brandes et Nick, 2011) (which use matrices). Having nodes laid out along one direction and time along another direction (Figure 2.2.4) is used in (Telea et Auber, 2008; Greilich *et al.*, 2009). A variant of this idea is used in (Rosvall et Bergstrom, 2010; Reda *et al.*, 2011), where nodes are first clustered, then clusters are laid out along one direction, and time along another, to show the merging and splitting of clusters over time. Finally, the 3D visualization (Figure 2.2.5) was used in (Brandes et Corman, 2003; Gaertler et Wagner, 2005). Burch et al. (Burch et Diehl, 2008; Burch *et al.*, 2010) have also proposed very original, but somewhat complicated, ways to visualize dynamic graphs that don't fit within Figure 2.2.

Note that three of the strategies (Figures 2.2.1, 2.2.2 and 2.2.5) can optionally use some kind of color or shape coding to indicate which nodes or edges are different in each snapshot, compared to the previous and/or next snapshots. *Difference highlighting* is a design dimension that is orthogonal to the possibilities in Figure 2.2, and can be combined with small multiples, animation, or 3D. This has been called a "difference map" (Archambault *et al.*, 2011b) or "difference layer" (Zaman *et al.*, 2011), and has been shown to be beneficial (Archambault *et al.*, 2011b).

In section 2.4 and onward, we will use the terms "small multiple" when no difference highlighting is used, "diff" for a static difference map that uses coloring to highlight differences between two (not necessarily consecutive) time slices, and "animation" for animations that also use difference highlighting. Our prototype supports all three of these representations. The reasons we will focus on these three representations are (1) they are the most studied approaches to date, compared to the others in Figure 2.2 ; (2) they are more scalable than arc diagram (Figure 2.2.4) or matrix-based approaches, which both require the height of the visualization to grow linearly with the number of nodes ; (3) matrix-based approaches make it harder to find paths (Ghoniem *et al.*, 2005) ; (4) the chosen techniques allow all kinds of changes to a network to be depicted, whereas the approaches in Figures 2.2.3, 2.2.4 have difficulty showing topological changes or changes in node position ; and (5) they avoid the occlusion and navigation problems of 3D (Figure 2.2.5).

2.3.2 Comparison of Animation and Small Multiples

Previous studies of visualizations that use animation have obtained mixed results. For example, Tversky *et al.* (2002) argue that animation is often misused or less effective than static representations. However, there are also examples of visualizations that were found to benefit from animation (e.g., Griffin *et al.* (2006); Heer et Robertson (2007); Chevalier *et al.* (2010)), though these did not involve dynamic graphs.

Three previous papers (Archambault *et al.*, 2011a; Farrugia et Quigley, 2011; Zaman *et al.*, 2011) have experimentally compared small multiples and animations for dynamic graphs. In two of these (Archambault *et al.*, 2011a; Farrugia et Quigley, 2011), small multiples were generally found to be superior to animation in terms of time to complete tasks. However, in both studies, when tasks required the user to examine specific nodes (e.g., "Which node has a degree that remains constant ?"), either the nodes to examine were highlighted with unique colors across all time slices (Archambault *et al.*, 2011a), or else all nodes in the graph were displayed with a mix of colors and shapes (Farrugia et Quigley, 2011). This makes it easier for the user to identify the node(s) of interest in each time slice, by simply looking for the appropriate color (or color and shape combination). A more general or realistic situation could have all nodes displayed with the same color and shape, in which case we could expect animation to yield more benefit when nodes are changing location from one time slice to the next. In partial support of this, Zaman *et al.* (2011) displayed nodes with the same color and shape across time slices. In their 2nd experiment, where nodes changed location, they found animation was superior to (static) "difference layers".

We also note that animation usually involves playback of a sequence of images, by pressing a key or a "play" button, at a speed determined before the animation starts. When the user, or more often the programmer, chooses the playback speed, there is a trade-off between how easy the animation is to understand, and how quickly one can view the complete sequence. We suspect that a more useful way to view an animation is by continuously dragging the pointing device (mouse), so that the user can continuously control the speed, and stop and reverse at any moment to review events of interest. This kind of interaction is often possible by dragging

along a time slider widget. However, all three studies may be biased against animation here. In (Archambault *et al.*, 2011a; Farrugia et Quigley, 2011), the time slider widget was small and therefore, by Fitts' law (MacKenzie, 1992), time consuming to acquire with the mouse, and in (Zaman *et al.*, 2011) there was no time slider at all. In real-world software, where users only casually view videos, navigation options are usually limited to a play/pause button and a small time slider. However, professional video editing software sometimes allow users to drag anywhere within a window to navigate within time, e.g., by using a special mouse button combination.

The experimental evaluation we present later in this paper is designed with realistic, expert use in mind. Therefore, in our work, nodes are not highlighted in a way that eases their identification across time slices, temporal navigation in animation is always done by dragging, and it can be performed almost anywhere in the main view.

2.3.3 Hybrid Visualizations

Previous work have explored the possibility of combining techniques to visualize a network at one moment in time (Henry *et al.*, 2007; Rufiange *et al.*, 2012). A paper (Hadlak *et al.*, 2011) also explored the usefulness of hybrids in the context of dynamic networks by nesting representations inside a different one. For instance, the structure of a graph is first depicted with a node-link diagram, and then other parts of the network are shown using different representations (e.g., complexity plot, matrix). However, this technique cannot be used to split a timeline into several parts, to try to benefit from using varying representations at different times in an evolving graph. Another work (Henry *et al.*, 2008) performed a user study to compare different node duplication techniques that could be used in the NodeTrix hybrid (Henry *et al.*, 2007) to improve the understanding of social networks, but did not involve dynamic graphs.

There are different ways to combine representations in information visualization in general, and these possibilities were discussed in Javed et Elmqvist (2012). While their classification is not focused on network visualizations, our work includes a juxtaposition mechanism, which is similarly used in multiple coordinated views (e.g., Roberts (2007); Federico *et al.* (2011);

Windhager *et al.* (2011); Zaman *et al.* (2011)). In contrast, the kind of hybrid approach we propose can reduce the need to switch between several views (e.g., difference view, animated view) by merging them and allows splitting a history into smaller parts. Multiple coordinated views can show several representations at once, but do not allow the more flexible and interactive mixing of visualizations as hybrids, and consume more screen space or make each representation smaller.

2.4 Taxonomy of Hybrid Visualizations with Focal and Context Regions

A graph can evolve over time and different visual techniques can be used to show these changes (such as small multiples, diff and animation). An interesting direction of research (also mentioned in Archambault *et al.* (2011a)) is to explore how these approaches can be mixed together. Imagine that we can interactively select different representations for each transition of a dynamic graph, depending on which one might be more appropriate in some context. Are the new combinations beneficial, and in which cases ? We use a taxonomy (shown in Figure 2.3) to organize possible combinations of visualization techniques for dynamic graphs.

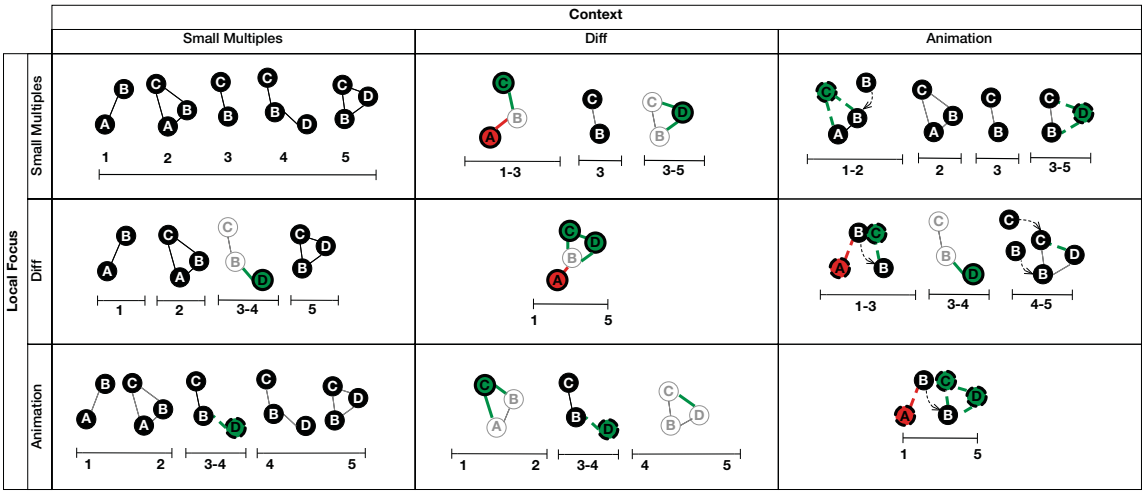

Figure 2.3 A taxonomy of different hybrid visualizations of the same dynamic graph. The context is the visualization technique used for surrounding time slices. Other time slices (i.e., the local focuses) can also be visualized using different techniques. The cells along the diagonal show "pure" (non-hybrid) techniques, whereas other cells show possible hybrids. Dashed lines are used to illustrate that some of the changes can be animated.

Taxonomies can help explore the design space of potentially useful combinations and can be constructed in different ways. A first step is to verify which techniques might be worth combining together. For example, node-link representations can help follow paths and are more familiar to users than matrices. However, matrices can be more scalable, even for dense networks. Since they are complementary in some ways, combining them may be useful (as shown in TreeMatrix (Rufiange *et al.*, 2012) and NodeTrix (Henry *et al.*, 2007)). Elastic Hierarchies (Zhao *et al.*, 2005) also aims to combine advantages of node-link diagrams and treemaps for visualizing trees. Similar reasoning lead us to combine visualizations to help understand dynamic graphs, since animation should help tracking moving nodes, whereas diff could enable quick identification of topological changes. We focused on investigating combinations that seemed more useful in practice (i.e., where the possible advantages of the combination were more clear).

In the taxonomy, we illustrate how different visualization techniques for dynamic graphs (i.e., small multiples, diff and animation) can be mixed in various ways. A small multiple represents a single "photo" of a dynamic network, taken at a specific moment in time. A difference map allow to highlight the changes between two small multiples, by combining them. Animations gradually interpolates changes made to a network, e.g., by moving and fading elements in and out. The combinations of these visualizations are expressed in terms of Focus+Context (as in (Zhao *et al.*, 2005; Rufiange *et al.*, 2012)) based on the idea that a main technique (the context) is used for a graph, but for some reason (e.g., possibly to optimize the screen space or better see movements), the user wants to use a different representation for a subgraph (the focus).

The timelines of a dynamic graph are shown in each cell of the taxonomy, along with possible visualizations for specific time steps and transitions. An evolving graph can thus be split in various ways and a different visual technique can be used for a region in "focus". For instance, in the middle column of the top row of the taxonomy, diffs are generally used to represent the dynamic graph, except at time step 3, where a small multiple is used instead. The opposite cell (in the left-most column of the middle row) illustrates another possible combination, i.e.,

insertion of a diff to highlight the changes between time steps 3 and 4, among several small multiples.

Another case (in the right-most column of the middle row) shows how a diff could be used instead of an animation in the middle transition of an evolving network. This could be useful if, for example, nodes do not move a lot for some period of time, and thus using an animation might not be the best approach in this case. In our prototype implementation, the user can interactively swap visualizations according to his/her own preferences (or heuristics), and for any time slices. Thus, it covers all the possibilities illustrated in our taxonomy, as well as similar ones, e.g., using two different focuses (diff and animation) in a context of small multiples.

2.5 Prototype

A video overview of our DiffAni prototype is available online [1]. Three kinds of tiles are supported : *small multiple* tiles (with no difference highlighting), *diff tiles* highlighting differences between two (not necessarily consecutive) time slices, and *animation tiles* (that also use difference highlighting). In difference highlighting, nodes and edges are shown in red or green if they are removed or added, respectively, across time slices. Currently, topological changes are the only kinds of changes visualized by the prototype. In addition, when the user place the mouse cursor over a node in a tile, the edges incident on that node, as well as the node's neighbors, are highlighted.

The user may use the mouse to zoom and pan over the sequence of tiles. The user can also drag sideways to navigate in time within animation tiles. Inside such a tile, the positions of nodes are interpolated based on the current position in time, and nodes and edges that appear or disappear are gradually faded in or out. We decided to use this scrubbing technique instead of having a "Play" button available to view animations with a fixed speed. Playback with a fixed speed is sometimes slower than necessary, causing the user to waste time, and at other times can be too fast, requiring the user to rewind the animation. On the other hand, forcing the user to scrub

1. http://ref.rufiange.com/diffani2013

manually means the user will always be navigating through time as fast as they can (or want), slowing down when they want or need to.

The "unified dragging" technique that we implemented (shown in Figure 2.4) uses the same sideways drag to pan and to navigate in time. This simplifies the input required from the user, obviating the need to switch between mouse buttons and/or between slider widgets. No matter what kind of tiles are in the sequence, navigation through time is always done the same way, and does not require first pointing at a small target with the mouse cursor (an action that could require significant time (MacKenzie, 1992)).

All of the combinations depicted in our taxonomy (Figure 2.3) are supported, and the mix of representations may be interactively changed by the user. As shown in Figures 2.1 and 2.5, the user may fluidly draw a stroke with their mouse to select one or several time slices (similar to drawing part of a lasso gesture), and then intersect their own ink trail (creating a pig-tail, inspired by Scriboli (Hinckley *et al.*, 2005)), popping up a menu allowing the selected tiles to be converted to a different representation. This quick gesture allows the user to create any mix of the three visual representations (i.e., small multiples, diff or animation). Whenever the user interactively changes the representation of time slices, an animated transition illustrates the conversion of tiles (this animated transition is not to be confused with the animation of the graph shown within an animation tile).

The design choices we made for the real experiment (e.g., the "unified dragging" technique) were based in part by the feedback we received in our pilot study.

2.6 Task-oriented Study

There are at least three theoretical sources of differences in performance that we can expect for the diff and animation techniques. First, with the diff technique, the user can compare two consecutive tiles with rapid eye movements, and even understand these two consecutive time slices by looking at the color highlighting within a *single* tile, whereas doing the same with the animation technique requires dragging with the mouse. Second, displaying N consecutive time slices with the animation technique is done with a single tile that is always visible, whereas the

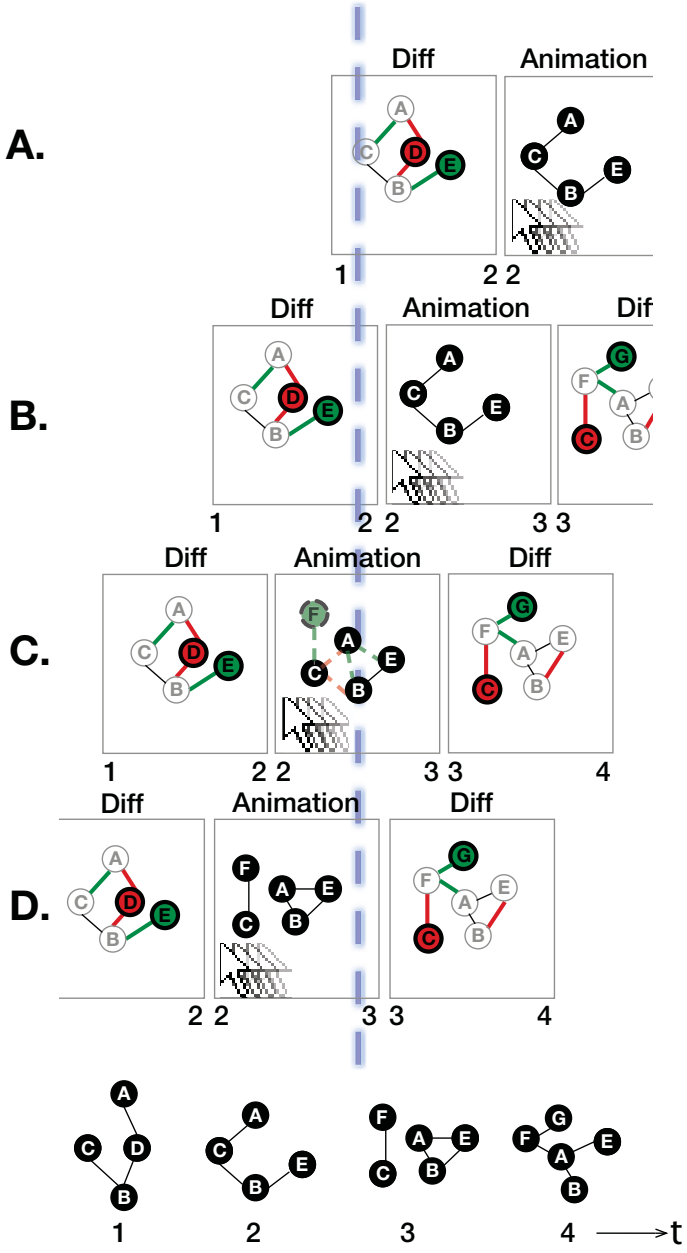

Figure 2.4 Illustration of the unified dragging mechanism implemented in our prototype. The blue dashed line indicates the current time (t) of a dynamic graph (also shown using four small multiples at the bottom). When a user drags the mouse inside small multiple and diff tiles (e.g., **A**, **B**), the sequence of tiles moves in the same direction as the dragging. Animation tiles (e.g., **C**, **D**) do not move in space to allow the user to travel in time instead. **A** : $t \approx 1$. All the elements inside the tiles are at their initial positions. **B** : $t \approx 2$. The animation tile remains unchanged, but the sequence of tiles has moved to the left. **C** : $t \approx 2.5$. The time cursor is between two time slices inside the animation tile, and thus the positions and the transparency of the nodes are interpolated. **D** : $t \approx 3$. At the end of an animation tile, the nodes are at their final locations.

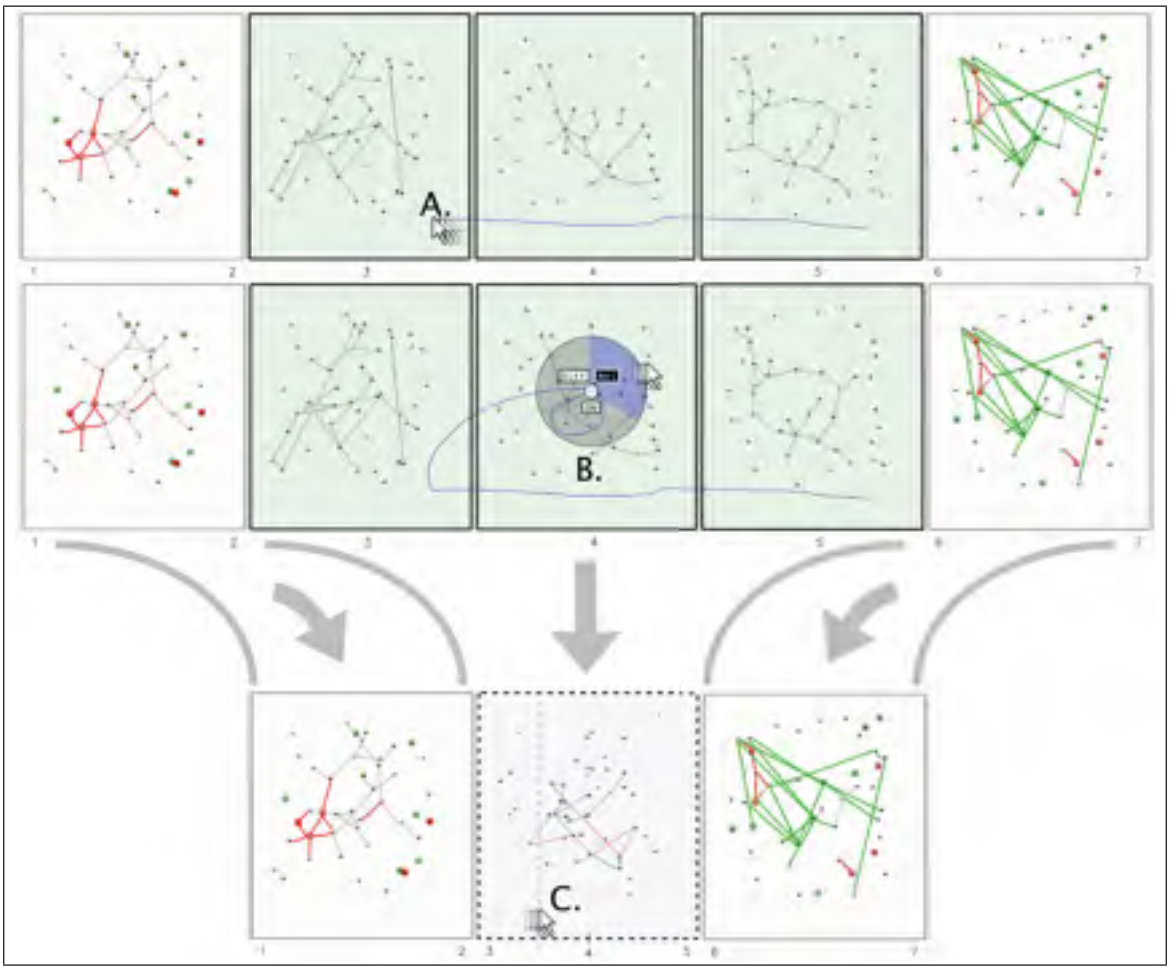

Figure 2.5 Interactively converting three small multiple tiles into a single animation tile. Initially (top of figure), we have a diff tile followed by three small multiple tiles followed by a diff tile. ***A :*** Dragging the mouse to select a contiguous set of tiles covering time slices 3-5. ***B :*** When the user intersects their own ink trail, a "pig-tail" is detected and a radial menu pops up, allowing the user to change the representation of the selected tiles. ***C :*** The previously selected small multiples are compressed into a single animation tile (this conversion is shown with a smoothly animated transition).

diff technique results in $N-1$ tiles that do not, generally, fit within the window, therefore requiring the user to perform some dragging to scroll through the tiles. Third, if there is significant movement of nodes from one time slice to the next, the diff technique could make it difficult to identify corresponding node positions in consecutive tiles, whereas the animation might make this clear by virtue of smoothly interpolating node positions. These three differences favor diff in the first case, and animation in the other two cases.

We cannot predict with certainty the net effect of these differences. However, we propose three hypotheses. First, we suspect that diff's advantage from fast eye movements will be strongest when there is little or no movement of nodes. Second, the relative performance of animation with respect to diff should improve when there is more movement of nodes. This second hypothesis is more speculative than the first, since performance with animation could also plausibly degrade, due to the user having to drag more *slowly* with faster moving nodes to monitor their individual changes. Finally, given the theoretical trade-offs between diff and animation, we suspect that the hybrid mixture of them in DiffAni can sometimes result in better performance than with either of the non-hybrid techniques.

Our hypotheses are thus as follows :

- **H1** : with greater movement of nodes across time slices, the performance of diff degrades.
- **H2** : with greater movement of nodes across time slices, the performance of animation with respect to diff improves.
- **H3** : the DiffAni hybrid visualization lead to better performance than with non-hybrid ("pure") diff or animation.

To test these hypotheses, we generated three datasets. All three datasets contain a mix of transitions involving either no movement or much movement, as illustrated in Figure 2.6. In the first dataset (A : low movement), the nodes do not move between time slices 1 and 3, then do move in the next two transitions, then cease moving. The second dataset (B : high movement) has a longer lasting stage of movement in the middle. The third dataset (C : phased) alternates between movement and stabilization. The datasets were generated by adding and removing random numbers of nodes and edges at each time slice using the parameters in Table 2.1. In normal usage, DiffAni allows users to select intervals of time and change the representation of a part of the dynamic network to small multiples, diff or animation. However, in our study, users were forced to visualize the network in one of three conditions : a "diff" condition which used only diff tiles, an "animation" condition which displayed the entire network in a single animation tile, and a "hybrid" condition which displayed low movement transitions with a diff

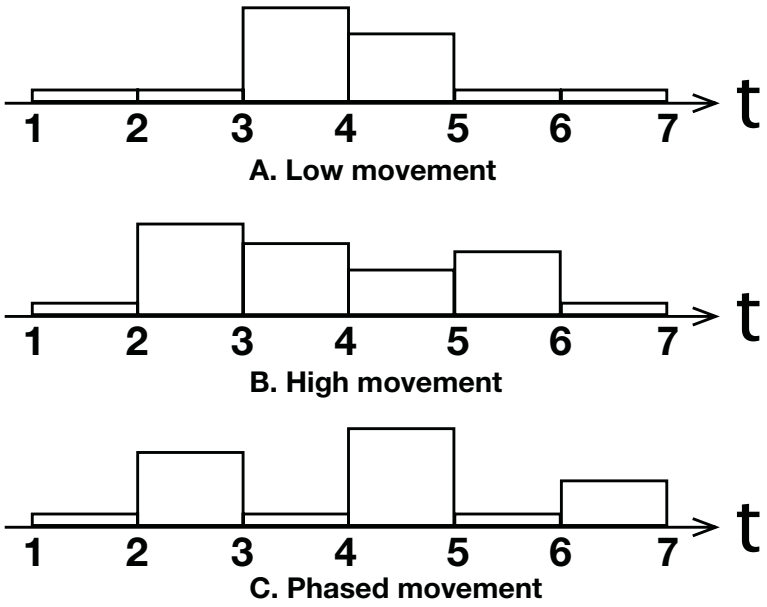

Figure 2.6 Schematic representation of the degree of movement of nodes in the three datasets used in our evaluation.

Tableau 2.1 Parameters used in the generation of the three datasets. Columns indicate average initial number of nodes and edges, average number of nodes added and removed in each transition, and average number of edges added and removed in each transition, respectively.

	$N(t=0)$	$E(t=0)$	ΔN+	ΔN-	ΔE+	ΔE-
Low	40.1	34.0	3.9	1.7	5.3	6.0
High	38.9	31.1	4.1	1.9	4.4	6.1
Phased	37.9	33.8	3.6	1.8	4.3	5.9

tile and high movement transitions with an animation tile. Users could not interactively change the representation of any tiles in the experiment.

The nodes in the datasets were positioned using a force-directed layout (Fruchterman et Reingold, 1991), computed for each time step of the dynamic graph. Differences in topology between the time slices naturally led to movements of nodes across time slices. Also, node positions were kept fixed for certain time steps, depending on the dataset (i.e., the transitions with a low degree of movement in Figure 2.6).

As shown in our taxonomy (Figure 2.3), a single diff tile can display the differences between two consecutive time slices or between two non-consecutive time slices. In the latter case, this serves to summarize several time slices in a single tile. However, in our experiment, the tasks required the user to examine every time slice and not just compare the first and last time slices. Therefore, the diff tiles always showed consecutive time slices. So, in the "diff" condition, there was a total of six diff tiles to show the seven time slices. In the "animation" condition, there was always a single animation tile, and in the "hybrid" condition, the number of tiles depended on the dataset. In particular, the hybrid representation had five tiles with low movement (four diffs and one animation), three tiles with high movement (two diffs and one animation), and six tiles with phased movement (three diffs and three animations).

Contrary to small multiples, diff can visualize changes between time steps, by combining two static representations and highlighting the differences. To place the nodes with the diff technique, we computed the averages of the starting and ending node positions for each transition of the evolving network. Another possible approach is to display several copies of the same nodes at once (e.g., Zaman *et al.* (2011)), but it can also make it more difficult to track individual nodes and consume more screen space.

We took inspiration from the set of questions used by Archambault *et al.* (2011a), and developed the following set of questions :

- **q1** : find which node is removed then reinserted most often over time.
- **q2** : find the node with the highest average degree over all time slices.
- **q3** : find the node whose degree never decreases.
- **q4** : find the path (chain of nodes) that is never disconnected in any time slice.

Comparing these tasks to Lee at al.'s taxonomy (Lee *et al.*, 2006), q1 is concerned with the appearance/disappearance of nodes over time, and cannot be classified in Lee at al's taxonomy which was not designed with dynamic graphs in mind. However, we find that q2 and q3 are Topology/Adjacency tasks, and q4 is a Topology/Connectivity task. Our set of questions is also comparable to that used by (Archambault *et al.*, 2011a), which included one Adjacency

task, one Connectivity task, and two tasks focusing on the appearance/disappearance of graph elements. All questions were multiple-choice with four candidate answers. To make these questions more difficult, additional nodes and edges were randomly inserted and removed over time. In addition, participants could not simply look at the last time slice to determine the answers. For instance, in task q4, paths were randomly disconnected then reconnected later on.

2.6.1 Experimental design

In the user study, to verify our research hypotheses (H1,H2,H3), we asked participants to perform four tasks using three different interfaces. Our motivation was to explore possible compromises in using the techniques, depending notably on node movements. In particular, we wanted to check whether our hybrid approach could benefit from using both animation (e.g., to track moving nodes) and diff techniques (e.g., the scrolling in time is unnecessary if nodes are stable).

The twelve participants (2 female and 10 male) were students in computer science, and the presented hybrid approach was new to them. We generated datasets to test all the conditions and tasks, and also made sure there was only one possible answer for each question and that it wasn't too difficult but still realistic, by adding a random level of noise. In the experiment, users first performed warmup trials for each technique (excluded from our final results) where they answered two questions about a 4th dataset that contained phased movements.

At the start of each trial, an instance of one of the questions was displayed in the main window, allowing the user to take the time to read and understand the question before pressing a "start" button. Next, the network data was displayed. In the first time slice of the network, the four candidate nodes (or pairs) were highlighted. However, these candidates were not highlighted in any other time slice, contrary to (Archambault *et al.*, 2011a).

The user could navigate by dragging until they determined the answer to the question, and entered the answer using radio buttons. If the first attempt by the user was wrong, they could retry up to two additional times, for a maximum of three attempts for each trial. Trials with

three unsuccessful attempts were counted as errors. Users were instructed to complete trials as quickly as possible, without errors.

In the final experiment, we used a within-subjects design with a total of

3 techniques (diff, animation, hybrid)
$\times$ 3 datasets (low movement, high movement, phased movement)
$\times$ 4 questions
$\times$ 12 users
= 432 trials in total.

The order of presentation of techniques was counterbalanced with a Latin-square design, and the ordering of datasets and questions was random for each participant. We controlled node movements in the datasets, based on our assumption that movement can play a role in deciding whether diff or animation should be used to visualize parts of dynamic networks. To test the effect of node movements in the experiment, node positions were sometimes kept fixed for specific time slices, causing non-optimal layouts (as illustrated in Figure 2.5 in time slices 3 and 6-7).

We ensured that the difficulty and complexity of the tasks were reasonable by doing a pilot study, that was performed prior to the final experiment. The pilot study included eight participants, and motivated certain changes in the prototype. For instance, in a previous version, the user used a traditional scrollbar widget to navigate in time within animation tiles (as in (Archambault *et al.*, 2011a)). Also, animations between consecutive time slices were displayed in three stages, to first show disappearing nodes and edges, then movements of nodes, then appearances of nodes and edges, following the staged animation approach of (Plaisant *et al.*, 2002). Both of these design choices seemed to penalize performance with animation. Furthermore, users seemed to be confused with using a scrollbar to navigate in animation tiles, while being able to drag anywhere to scroll through diff tiles. In the staged animations, some participants had difficulties distinguishing between the different slicings that were used in the timeline, i.e., slices to separate time steps, and also slices for each stage of an animation (placed between two

time steps). Therefore, for the full study, we adopted the unified approach in Figure 2.4, where dragging anywhere serves to navigate both kinds of tiles. Our pilot study also tested a condition where all time slices were shown with small multiple tiles (without coloring of differences). However, since these small multiples were clearly inferior to the other techniques, they were excluded from the final experiment.

2.7 Results

In this section, we present the results of our experiment. The total time and error rates are shown in Tables 2.2 and 2.3.

Tableau 2.2 Average duration of task trials for each technique (in seconds), and grouped by movement type. These include the time spent on 2nd and 3rd attempts when the user's 1st attempt was wrong.

	Low	High	Phased
Diff	135	140	128
Animation	117	147	**106**
Hybrid	**104**	**123**	128

Tableau 2.3 Average number of attempts per trial (for all tasks). In parentheses are error rates, where a trial is considered an error if the user's 3rd and last attempt is still wrong.

	Low	High	Phased
Diff	1.13 (0.0%)	1.09 (6.4%)	1.00 (0.0%)
Animation	1.11 (2.1%)	1.15 (0.0%)	1.09 (2.1%)
Hybrid	1.09 (2.1%)	1.11 (2.1%)	1.15 (0.0%)

Shapiro-Francia tests on total times revealed them to not be normally distributed, hence Friedman tests were used to check for significant differences between visualization techniques for the different tasks and movement conditions. Over all movement types, the hybrid was found to

be (weakly) statistically better than diff ($p < 0.10$). Examining only the average times, hybrid has the best total times in two of the movement conditions (indicated in bold in Table 2.2).

We also checked whether some visualization techniques were harder (i.e., required more attempts) depending on the movement condition. Table 2.3 shows that the error rates (after 3 attempts) were below 5% in almost all cases, indicating that users were able to complete the tasks.

The participants made fewer attempts on average with the hybrid in the low movement condition, while the number of attempts was lower with diff in the other cases. However, with high movement, diff yielded a higher error rate than the other techniques. As for phased movement, diff resulted in no mistakes. Also, no significant differences were found ($p > 0.10$). These results indicate that the techniques were not very different from each other, in terms of number of attempts. Moreover, these results suggest that the hybrid approach was not more difficult to use than the non-hybrid techniques, although it required skills in both visualizations. Table 2.4 shows the total times broken down by task.

Tableau 2.4 Results for all interfaces, tasks and movements (times are in seconds). Within each task and movement combination, the minimum time is shown in bold (or two numbers are in bold, if they are not significantly different from each other). Two stars indicate the bold number(s) is (are) significantly ($p < 5\%$) smaller than the non-bold number(s), whereas one star indicates weak significance ($p < 10\%$).

Low movement	Task 1	Task 2	Task 3	Task 4
Diff	132	251	**45***	109
Animation	**107**	**219***	**54**	86
Hybrid	120	**160***	55	**80**
High movement	Task 1	Task 2	Task 3	Task 4
Diff	**123**	253	**63**	**120***
Animation	162	232	68	126
Hybrid	**134***	**206**	**49***	**100***
Phased movement	Task 1	Task 2	Task 3	Task 4
Diff	100	256	**34***	121
Animation	**96***	**198***	53	**76**
Hybrid	143	**207****	58	104

Regarding hypothesis H1, examining the averages, diff performed worse in the high movement condition than the low movement condition in three of the four tasks, indicating that collecting more data might confirm H1.

Concerning H2, when we compare low movement and high movement, we find the change in time for animation is always worse than the change in time for diff (i.e., the performance of animation decreases more than diff with higher movement). This contradicts H2. This may be because users had to drag very slowly in the high movement condition to be able to track node movements. Figure 2.7 shows the fraction of time spent dragging in each condition. As can be seen, the percentage of time spent dragging is higher with animation, compared to diff, in every condition. H3 was partially confirmed : in the low movement condition, the hybrid was (weakly) significantly faster than diff for task 2 ($p < 0.10$), and it also had the lowest average for task 4. In the high movement condition, the hybrid was (weakly) significantly faster than animation for tasks 1, 3 and 4, and it was the best technique in terms of average time for three of the four tasks. With phased movement, DiffAni was significantly better than diff for task 2 ($p < 0.05$).

Comparing diff and animation in the low and high movement conditions, we find that diff seems less sensitive to movement, whereas performance with animation changes more between movement conditions. Interestingly, Purchase et Samra (2008) found that user performance with animation varied non-linearly with the degree of preservation of mental map (which varies, roughly speaking, inversely with the degree of motion of nodes across time steps). These two results indicate that performance with animation varies in a complicated way, making it difficult for designers to choose a prescribed representation.

Analyzing the data by task, we notice that tasks 1 and 2 took more time than tasks 3 and 4. This is not surprising, since users could sometimes complete tasks 3 and 4 before examining all the candidate answers (e.g., in task 3, if a user noticed that a candidate node's degree never decreased, he/she could try to immediately answer the question without examining the other candidates). Tasks 1 and 2, however, required that all the candidates be examined before answering. Furthermore, between tasks 1 and 2, task 2 required a more complicated mental cal-

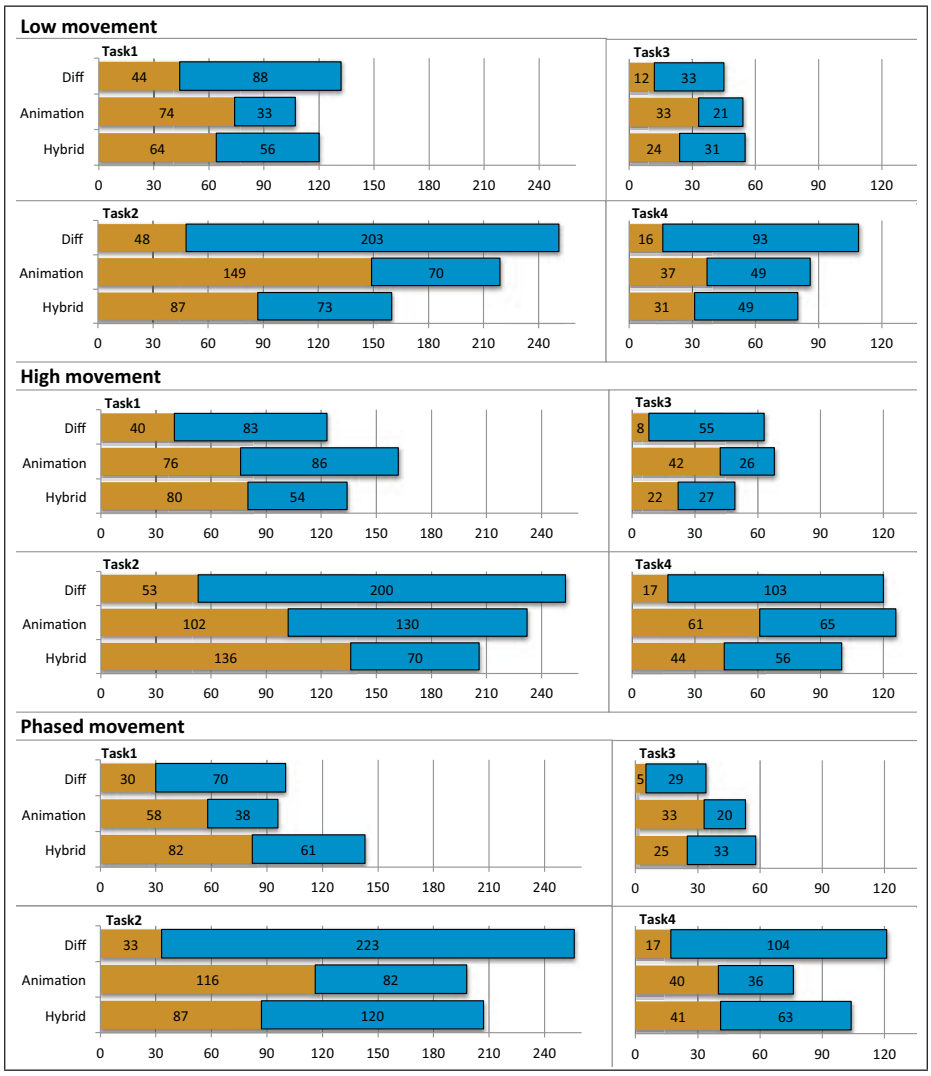

Figure 2.7 Distribution of the time spent performing tasks using the visualization techniques, depending on the degree of movement (times are in seconds). The orange-colored bars indicate the participants were dragging in space (e.g., diff) or in time (e.g., animation), while the blue bars show the non-dragging time.

culation, and not surprisingly took the most time. Comparing tasks 3 and 4, task 3 was simpler and took less time than task 4 (which required examining entire paths of nodes).

In the first task (q1), animation was the fastest approach in the low movement condition, possibly because it facilitated the tracking of removed nodes (in Figure 2.7, the non-dragging time is lower). However, the performance of animation decreased with high movement. We suspect this is due to the fact that animation may not be the best technique to track new nodes, and

even more so with increased movement, since they can appear anywhere. In this case, it may be faster to simply look for highlighted differences rather than scrolling in time with animation. Probably because of this more complicated trade-off, the hybrid approach performed average.

In the second task (q2), the hybrid visualization was the fastest technique with low and high movement. Looking at Figure 2.7, we can see it achieved significant gains in terms of non-dragging time (shown in blue). We suspect the unified dragging might have facilitated the tracking of nodes across representations. Since an animation is only used when there is movement, it is easier to track the nodes over time. Also, in such a hybrid representation, the starting and ending steps of a animated transition matches the previous and next node positions shown with a difference map (as shown in Figure 2.4).

In the third task (q3), participants were faster using diff, except in the high movement condition. Scrolling in time with animation may have been generally less effective for this task, compared with finding highlighted changes. We suspect that, since the degrees of some nodes were increasing over time, it might have made them a bit easier to locate. However, the tracking of nodes across time slices was probably more difficult with diff, especially with high movement. The hybrid approach, which used animation only sparingly and also integrated the unified dragging technique, performed better in this case.

In the last task (q4), the hybrid approach was generally faster, although it was slower than animation in the phased movement condition, possibly because of the mental effort required to switch representations very often. In this case, participants commented that animation generally helped them track nodes and chains of nodes across time slices. These results suggests that participants preferred animation to track nodes, but the usage of diff was also beneficial when nodes did not move much.

Examining average times for hybrid, within each task and across all movement conditions, we noticed that phased movement always yields the worst performance. For example, within task 1, hybrid took 143 seconds on average with phased movement, vs. 120 and 134 seconds in the

other movement conditions. This indicates that the phased movement condition resulted in too many changes in the hybrid visualization ; this is also corroborated by user feedback.

2.7.1 User feedback

In our user study, participants experimented with several approaches. In addition to collecting quantitative data (e.g., number of attempts, task durations, error rates), we also used five-point scales to evaluate their impressions.

The participants generally felt comfortable using the visualization techniques. In particular, animation and diff were rated as very easy to use (4.1 and 4.0, respectively), while the hybrid, perhaps because of the surprising interface and multiple representations, was evaluated at 3.6. However, it is interesting that participants also rated the hybrid as the most useful approach to perform tasks (4.3 vs. 3.9 for diff, and 4.0 for animation). We believe this correlates with other feedback we received that animation clearly helped track nodes efficiently, but only if used sparingly (as with the hybrid).

Several participants preferred our hybrid approach, arguing that tracking nodes was easier and that the use of animation was very useful and generally faster (P4,P5,P8,P9,P11-P13). Users further explained that they liked that the hybrid only uses an animation if a node actually moves over time (P4,P8,P11). However, some participants also said the performance of the hybrid decreased with alternating phases, mostly because they had to adapt to changing representations (P5,P8,P9). A few users felt that animation was not always the best technique, because they had to scroll in time too much (P7,P12), although another (P2) argued it was better to scroll in time in one big tile than across several tiles (sometimes causing a loss of context).

Participants also had suggestions for improvements, such as reducing cluttering (P1,P2,P6,P12) or highlighting nodes in several time slices (P4,P6,P9). They also believe that it could be useful to display tooltips showing, e.g., the number of neighbors of a node (P7,P9) or allow selecting nodes from a drop-down list (P4,P5,P9).

2.7.2 Applications

In this paper, we have shown that a hybrid technique can potentially help analyzing dynamic networks, but there are also practical uses of these combinations. In fact, to fully use hybrid approaches such as ours, one has first to construct a hybrid representation of the data and then analyze the resulting visualization. We focused on the latter case in our study. However, since users could benefit from mixing techniques (i.e., diff and animation) in some cases in our experiment, we propose a heuristic to construct hybrid representations of dynamic networks (illustrated in Figure 2.8).

First, a metric to evaluate the degree of node movements has to be computed for each transition (e.g., the average distance between time steps). Second, if there is no or little movement in a transition, diff can be used. Otherwise, animation was generally found to be more suitable. Also, if the same visual techniques are used for several consecutives transitions, they should be merged together, to avoid frequent switching in representations. Since small multiples

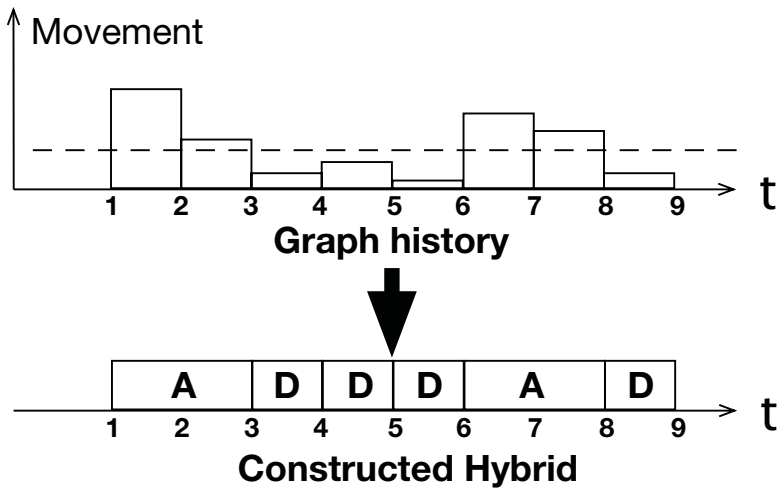

Figure 2.8 A method to construct potential hybrids using our prototype. In this case, appropriate visualizations are chosen based on the degree of movement of nodes in the evolving network.

performed significantly worse in our pilot study, it was not used in the real experiment. We

therefore suggest that it should be only used for a small number of time slices, and also when the differences in transitions does not really matter.

In the spirit of previous work on hybrid visualizations (e.g., Henry *et al.* (2007); Rufiange *et al.* (2012)), a more suitable representation can be flexibly used for different parts of a dynamic network using our approach. For instance, in software engineering, refactorings can cause more instabilities (e.g., new modules are added, moved or new connections are made) at the beginning of a project, and also once in a while in the software change history when the design is reworked (Jacobson *et al.*, 1999). However, bug fixes might not induce a lot of changes in terms of connectivities. Thus, the presented hybrid technique could help a software engineer understand how a design evolves over time, by using appropriate visualizations for different periods of time. As a software is usually developed over many years, this could allow engineers to browse large software histories faster.

There are also other possible applications of our hybrid approach that will be explored in future work. For instance, in a cell phone network, there are less movements at night, and depending on the time of the day, an appropriate visualization could be used. Also, in social networks, some connections do not change a lot while others can be more unstable (e.g., childhood friends vs. part time employees). Although we presented a heuristic to construct hybrids, other mapping functions to try to select appropriate representations over time should be investigated.

2.7.3 Limitations

We made choices in designing our experiment and prototype that we discuss in this subsection. We implemented a Fruchterman-Reingold algorithm (Fruchterman et Reingold, 1991) in our prototype, rather than possibly more complex strategies with unclear outcomes (Purchase et Samra, 2008). Also, more advanced layout algorithms are not always included in compatible software libraries or require expert knowledge. In our experiment, because of our choice of layout algorithm, we could not reproduce the exact same effects of mental map preservation which have been used in certain previous studies (e.g., Purchase et Samra (2008); Purchase *et al.* (2006)), however we could indirectly vary mental map preservation, by varying the to-

pology of the graph (i.e., adding and removing nodes and edges), which caused movement of nodes.

Our hybrid method relies on generic visualizations that can be used for any network. Also, the dynamic graphs in our datasets were randomized and had roughly between 35 and 70 nodes over seven time slices, similarly to previous studies (e.g., Archambault *et al.* (2011a,b)). However, user studies will be required to explore the possible effect of network density or the number of time slices on the performance of participants for more tasks and application domains.

In our experiment, the tasks chosen for the evaluation are based on existing taxonomies (e.g., Lee *et al.* (2006)), but do not cover higher level changes (e.g., clustering, overview) nor attribute changes. We did not allow the highlighting of nodes over several time slices in our user study, to focus on comparing the visualization techniques themselves. Also, some tasks do not benefit from highlighting (e.g., finding a node, among many possibilities, that evolve according to some pattern).

We suspect a few uncontrollable factors could have contributed to reduce the statistical power of our results in addition to the small sample size. For instance, to use the hybrid fully, users needed to master two techniques, although some did not like or were less efficient with animation (while others liked it more than diff). To be fair, we let the participants practice the same amount of time using each technique, although users may have their own preferences and skills.

We used a fixed hybrid representation and did not allow the user to interactively construct hybrids in the experiment to limit the variability of our results (as done similarly in e.g., Archambault *et al.* (2011a)). We focused on finding benefits on combining representations, and thus mixed typical and simple non-hybrid visualizations (e.g., diff, animation). However, the combination with other unexplored techniques (e.g., sparklines) could be beneficial. Also, the comparison of hybrid representations with other non-hybrids (such as multiple coordinated views (Roberts, 2007)) should be explored in the future.

2.8 Conclusions and Future Directions

We have presented a hybrid visualization of dynamic graphs that allows the visual representation of the data to be varied across time. Multiple consecutive time slices in the data can be collapsed into a single tile displaying either an animation or a static difference map, enabling the user to choose trade-offs between space and time for different portions of the data.

We also proposed a taxonomy of visualizations of dynamic graphs (Figure 2.2) and a taxonomy of hybrid visualizations (Figure 2.3). Our prototype implementation allows users to flexibly explore these combinations.

Our experimental evaluation showed that the performance of different visualization techniques does not vary in a simple way, but that nevertheless the hybrid visualization can outperform other techniques in certain conditions, indicating that hybrids can result in a better trade-off than non-hybrid alternatives. We therefore recommend that hybrid visualizations allow users to interactively control the mixture of representations used for the data.

The user study differed from previous work in two ways that make it relevant for real-world expert use scenarios : first, the candidate nodes for tasks were not highlighted across all time slices, and second, navigation in time and space was performed with the same kind of mouse dragging (Figure 2.4) rather than requiring the user to use a small slider widget or play animations with a fixed speed.

One issue that could be investigated in future work is the reason why animation performed more poorly than expected, contradicting our hypothesis H2. We suspect that users may have been hindered by having to drag slowly during animations to be able to understand rapid node movements (and because of the added noise in the datasets). Future prototypes may improve performance by displaying “smeared out trails” or motion blur effects to make the movements of nodes clearer in fewer frames, and/or non-linearly vary the mouse cursor gain according to the speed of movements, to make it easier for the user to quickly understand transitions between time slices.

Another interesting research direction concerns the construction of hybrids. Although we presented a heuristic to design hybrids for dynamic networks, there is a need for more guidelines to determine how visualizations should be mixed in some context. Users that tried a few iterations of our prototype were generally faster with the hybrid, and thus it could be useful to study the learning curves of users to understand how to better exploit hybrid visualizations. Also, the interactive process of assembling hybrids itself could lead to discovering insights. Furthermore, alternative hybrid representations should be explored and evaluated to study, for instance, the effect of using different layout algorithms and interaction techniques.

CHAPITRE 3

ARTICLE 3. VISUALIZING PROTECTED VARIATIONS IN EVOLVING SOFTWARE DESIGNS

Sébastien Rufiange[1] and Christopher Fuhrman[1]
[1]Department of Software and IT Engineering, École de technologie supérieure,
1100, rue Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
sebastien@rufiange.com, christopher.fuhrman@etsmtl.ca
Article soumis à la revue "Journal of Systems and Software" le 7 avril 2013.

3.1 Abstract

Identifying and tracking evolving software structures at a design level is not an easy task. Although there are ways to visualize this information statically, we need more methods to analyze evolving software design elements. In this paper, we present a new visual approach to identify variability zones in a software design and explore how they evolve over time. To explore the usefulness of our approach, we did a user study in which participants had to browse software histories and find visual patterns of interest about a design. Most participants were able to find interesting observations and found our approach intuitive and useful. We present a number of design aspects that were observed by participants and the authors using our tool on different real-world open-source projects.

Keywords : information hiding, software design, software evolution, software visualization.

3.2 Introduction

Changes in a software project happen for different reasons. There can be bug-fixes to respect quality attributes such as reliability, security or performance. There can be changes due to the uncertainty that is inherent to early phases of risk-driven iterative projects. There can be

additional functional requirements. There could be a need to run the software on a different platform or on a different operating system. An important characteristic of an enduring software design is its ability to handle change over time.

Information hiding (Parnas, 1971) is a core principle in structured and object-oriented design. Designs that apply information hiding aim to hide parts of the software that are likely to change in order to reduce the impact of that potential change on other modules. Information hiding favors designs that have loose coupling to the elements that are potentially unstable.

There are different strategies to achieve information hiding. *Encapsulation* has been defined as "building a capsule, [...] a conceptual barrier around some collection of things" (Wirfs-Brock *et al.*, 1990). Encapsulation implies that a designer explicitly specifies the boundary and what is visible to the outside. Programming environments typically offer mechanisms of *access control* of capsules to specify and enforce what is hidden and what is visible. A simple example is a Java class that has private members with public methods. Access-control encapsulation can also be used at the package level in Java, such that classes in one package are invisible to classes outside of that package.

A related idea is the *open/closed principle* (Meyer, 1988), which suggest that new features in a software should be implemented by extensions (e.g., adding new classes and methods to a sub-class) rather than modifications, to reduce the impacts on client modules that depend on existing features. Similarly, the idea of *protected variations* (Cockburn, 1996; Larman, 2001, 2005) seeks to isolate what is change-prone (or unstable) behind an intentionally stable interface. Polymorphism or composition can then be used to define varying implementations while the clients only access the interface. Larman (2001) mentioned that the open/closed principle and protected variations are essentially equivalent to the more generic and fundamental information hiding principle. McConnell (2004) proposed the *iceberg metaphor* as shown in Figure 3.1. In terms of the open/closed principle, a module is said to be "closed to modification" yet "open to extension". These relationships are always with respect to a client (or set of clients) that should not be affected by the extensions. This defines a frame of reference relative to client classes.

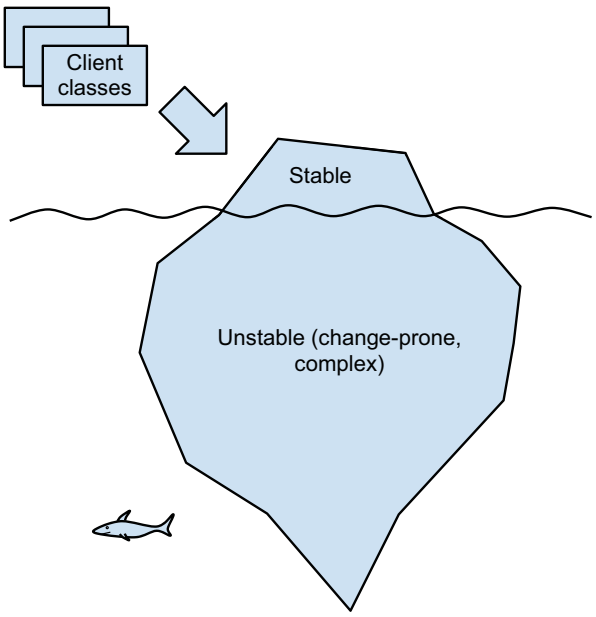

Figure 3.1 Iceberg metaphor for information hiding proposed by McConnell (2004). Client classes should not see that which is considered unstable or change-prone.

In practice, popular object-oriented design patterns (Gamma *et al.*, 1994) strive to make designs more tolerant to changes. Many of these patterns make use of protected variations to protect respective client classes from extensions or modifications to the software. These dimensions of extension are intentional, with structures that use polymorphism (e.g., in Strategy and Observer) or composition (e.g., in Facade, Iterator and Proxy). However, despite the intentional dimensions for extension, these patterns are not explicit in their definitions about what is hidden or visible to the clients. Protected variations implies that clients only use the stable interface ; the information hiding principle implies they should not know or see the extension classes. Ideally, a designer could specify this with access control. However, this is impractical in some environments with traditional access control mechanisms. Extension (hidden) classes could be in various different packages, scattered somewhat arbitrarily throughout a design. It is therefore challenging to determine the boundaries of the *capsule* (i.e., the iceberg).

3.2.1 Controlling access to unstable elements

Grand (2002) proposed the Interface pattern which is indeed a realization of protected variations. The pattern proposes a solution that keeps "client classes independent of specific data-and-service-providing classes" such that changes to the latter classes will not affect the clients. Figure 3.2 illustrates the pattern implemented in the context of a package, such that package access control prevents clients from seeing the implementation (Service) classes. This access

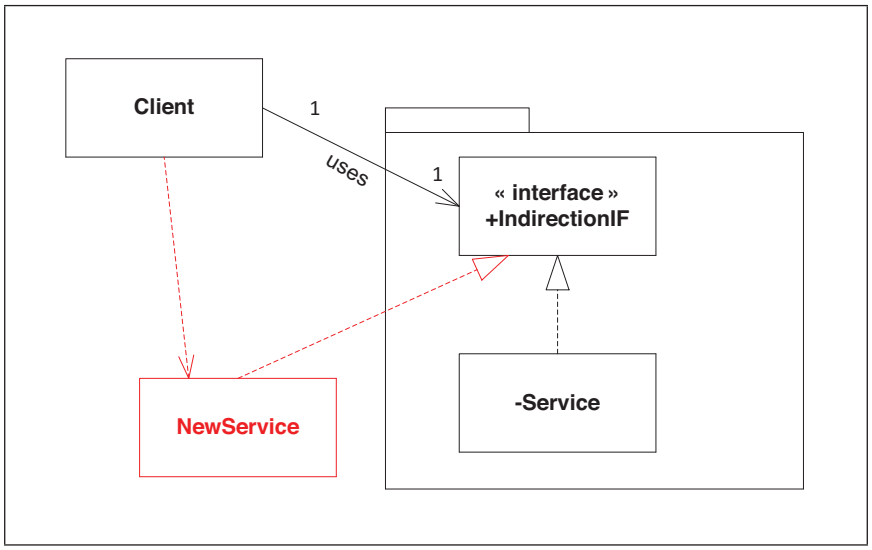

Figure 3.2 Interface pattern in Java (Grand, 2002). Package-level access controls prevent coupling to implementations inside the package only.

control strategy works well provided all the implementations can be constrained within a package. Consider, however, an extension to this design where some NewService is added outside the package. There is no standard way of preventing clients from accessing this NewService directly.

Since strict access control beyond the class level is not commonly used by developers, the principles of "structure hiding" and "information restriction" are not always respected in practice. In fact, the *Law of Demeter* (LoD) essentially says that developers of client classes should be more conservative about using the information they can get from a class (Lieberherr *et al.*, 1988). Also known as "Don't talk to strangers," the LoD could be thought of as follows : every

class that a client class accesses *could be* a Facade [1]. By not accessing the "strangers" behind the Facade, a client class is protecting itself from potential variations of those classes.

Because traditional private-public access control mechanisms in languages are often not applied at architectural levels, there exist ad-hoc solutions. One such solution is the "Explicit Extension Rule" in Java Eclipse Plugins, which is a convention to name hidden, change-prone packages as ".internal." (Gamma et Beck, 2003). The access control to these packages is not enforced by a compiler, but environments such as Eclipse give warnings or errors when code is built [2] if these access control conventions are not followed. Similarly, manifests in OSGi bundles (Hall *et al.*, 2011) can allow defining modular components with access control enforced within the Eclipse Equinox reference implementation.

Some language-specific encapsulation mechanisms exist beyond the class and package level. The "internal" access modifier in C# (Microsoft, 2012) limits visibility of members to files in the same *assembly*, and there is a planned extension to Java to support modules (Reinhold, 2011). However, even with a broader scope of access control, it remains a challenge to assure that the type of extension proposed outside the scope of an arbitrary "module", as depicted by the NewService class in Figure 3.2, that is visible to the Client. The problem stems from the fact that modules are specified in a top-down way, rather than being identified dynamically based on coupling.

3.2.2 Tracking protected variations over time

Differences in the software development process affect the way that designs evolve, and in turn how encapsulation might be specified during the process. As stated by (Parnas, 1971), it is the designer who decides where the boundary is between what it hidden and what is private. This is traditionally a top-down strategy of design, because important decisions are made *before* client programmers are exposed to the programming interfaces. At the class and package level, things are straightforward : a designer proposes a software class and specifies which parts will

1. or the client-facing class of any of the patterns that hide change-prone or complex structures, e.g., Proxy, Iterator, Factory.
2. http://help.eclipse.org

be visible to the outside and which parts will be hidden. However, designs involving a lot of composition of reusable libraries are said to be *emergent* (McConnell, 2004). It may not be clear at the higher levels where to apply information hiding ; if it is applied too conservatively, it might restrict freedom in bottom-up design. Furthermore, design patterns are often applied in groups (Buschmann *et al.*, 1996). The unstable elements of one pattern might be clients of stable elements of other patterns.

Therefore, specifying access control at the architectural level is arguably more challenging. Again, some solutions exist : the Layers pattern and the Model-View-Controller pattern (Reenskaug, 1979) are both examples of where there is a *convention* of access control. That is, certain lower-level layers (i.e., the Model) should not be coupled to ("see") upper-level layers (i.e., the View).

Keeping track of coupling over the evolution of a project is an important concern for software architects, but so is the tracking of the dimensions of variability. Once a design with protected variations is specified (regardless of access control conventions), several questions remain : (1) does the protected-variations dimension of the design serve a purpose (are new variations implemented) ? (2) do the "stable" parts get re-used by more and more clients (are they "popular" or "useful" abstractions) ? and (3) are the coupling conventions respected (are the clients decoupled from the change-prone parts "below the water" of the iceberg metaphor) ?

As software grows in complexity, the changes to it become more difficult and can cause *architectural erosion and drift* (Perry et Wolf, 1992). In terms of protected variations, this implies that as software is added, the protected variations structures could no longer be valid. An example (Figure 3.2) of this theoretical drift is that a new class is added which is both a client to the stable part of the protected variations but for some reason is also coupled to its extensions.

3.2.3 Visualizing changes in software designs

Interactive visualization techniques and visual analytics can help users discover insights in (possibly dynamic) data, such as unexpected patterns, anomalies or relationships (Thomas et Cook, 2006). Because of software's intangible nature and its tendency to be complex, graphic

visualizations have been proposed to better understand software design. Software visualization can cover three main aspects : structure, behavior and evolution (Diehl, 2007). Visualization of evolution is essentially doing a visual analysis of software histories, of which there are three kinds : visualization of metrics, visualization of structural information, and discovery of recurring patterns from the software history using data-mining and visual data-mining techniques. Our approach combines aspects of all of these kinds to analyze the *evolution of structures*, particularly the structures associated with micro-architectural protected variations, where it is more difficult to specify and enforce encapsulation.

Despite a good visualization, a software project's complexity can still overwhelm the observer. Many visualization techniques tend to display the entire structure of the software, rather than the perspectives derived by encapsulation (i.e., the iceberg metaphor doesn't apply). By displaying only coupling related to protected variations, for example, a tool could potentially help a developer focus on those aspects, without being distracted by coupling that is unrelated. Visual tools have the potential to help discover unknown interesting patterns by engaging a user in an interactive exploration of a software. To achieve this, the "details on demand" principle in information visualization (Shneiderman, 1996) can be applied, to allow the user to get an overview first, and filter the data based on some classes, types of object, and instances of particular interest or evolving in certain ways.

We propose in this article a visualization tool, applied in the software engineering domain, that presents the elements of a software design to the developer in terms of the frames of reference established by the open/closed principle. To account for evolution in the design, our interface allows visualizing multiple versions of a software in an Subversion repository. We illustrate how our novel approach can help to (1) find periods of interest in a software change history, (2) find design structures in real projects and (3) track and characterize evolving design structures.

The rest of this article is organized as follows. Section 3.3 presents the variability concepts on which our visualization tool is designed. Section 3.4 presents the visualization tool itself, and then authors' observations regarding protected variations in open-source projects. Section 3.5 presents the task-based user study that was done to show the usefulness of the tool in the

software design community. Section 3.6 discusses all of our results and limitations before section 3.7 explores similar work, comparing and contrasting it with this article's contributions. Finally, section 3.8 draws conclusions and discusses ideas for future work.

3.3 Variability Zones

This section describes the key concept of *variability zones* as well as how they evolve over time in a software project. As mentioned in the introduction, there are several mechanisms commonly used to achieve protected variations. One popular strategy in the Java language is the Interface pattern (Grand, 2002), and it illustrates well the elements of the variability zone concept that are fundamental to our approach.

Assuming that a Java interface is an information hiding mechanism that can be used to hide variations from some external clients, protected variations can only be achieved if the clients are coupled only to the interface and are decoupled from concrete implementations. As shown in Figure 3.2, a static package (or arbitrary capsule) could be insufficient to capture the future extensions of implementations of a service, since a developer could create a NewService implementing the public IndirectionIF interface.

A *stability point* (mapped to a Java interface) thus defines a **variability zone**, which is **the set of elements implementing** (e.g., abstract classes, concrete classes) **or extending the interface** (e.g., derived interfaces), as illustrated in Figure 3.3. This definition can also cover one or several levels, to include the concrete elements implementing derived interfaces of the stability point. At level 0, the variability zone only includes the concrete elements (i.e., implementations and extensions) of the stability point's interface itself. At level 1, the variability zone also covers the concrete elements of the implementations and extensions of the stability point's interface, and so on.

Variability zones also have properties that we can collect over time. N refers to the actual number of elements inside a variability zone (e.g., classes), and these are extending a stability point in some way. The number of distinct clients coupled to the stability point is known as "stable coupling" (counted by S). Alternatively, instances where clients depend on concrete

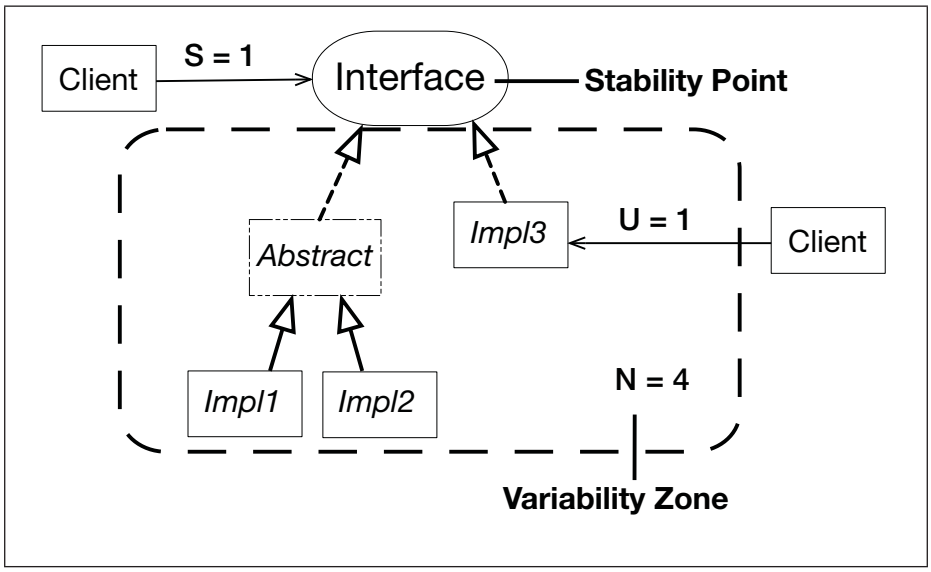

Figure 3.3 Illustration of a variability zone and its properties, using UML notation. Property N : The number of elements inside the boundaries of a variability zone. Property S : The number of elements coupled to the stability point (stable coupling). Property U : The number of elements coupled to an element inside the variability zone (unstable coupling).

implementations defined inside a variability zone represent “unstable coupling” (counted by U). Unstable coupling implies that clients are not protected from changes inside the variability zone. Property N is similar but more specific than the NOC (number of children) metric from (Chidamber et Kemerer, 1994). In theory, N is the number of children that implement a stability point, where each child is part of the hierarchy of the stability point. In practice, the depth of the hierarchy for this counting (i.e., the number of levels to include in a variability zone) can be changed in the counting algorithm.

3.3.1 Evolution of Variability Zones

Variability zones can evolve over time in different ways during the history of a software project. Figure 3.4 illustrates several hypothetical possibilities considered in our approach, and how they would be observed in terms of structure and the properties of N, S and U.

The fluctuations of a variability zone can also be viewed graphically in terms of the counts N, S and U. They can be interpreted in terms of the information hiding principle. For example, if the

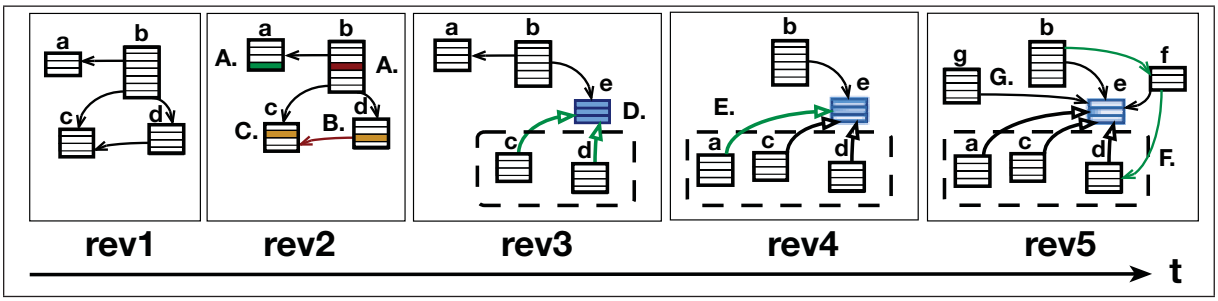

Figure 3.4 Change history of an hypothetical evolving design to illustrate the typical changes happening in general and in a variability zone and how it is visualized in our approach. **A**. Method or classes are added (in green) or removed (in red). **B**. The implementation of a method is changed and a dependency is removed. **C**. A method signature (e.g., a parameter is added) is changed. **D**. A new stability point *e* is introduced that *c* and *d* now implement. **E**. The size of the variability zone increases (N) as *a* is added. **F**. Unstable coupling increases (U) as a new intermediary class *f* links *b* to concrete implementation *d*. **G**. Stable coupling increases (S) as *g* becomes a client of *e*.

boundaries of a variability zone increase significantly while the level of unstable relationships remains constant, it suggests that the instabilities of this part of the design do not increase around the stability point. Indeed, no new external clients are coupled (for the period of time analyzed) to the concrete implementations that were added. Figure 3.5 illustrates a few other possible cases in terms of the evolution of a variability zone. We now describe an hypothetical

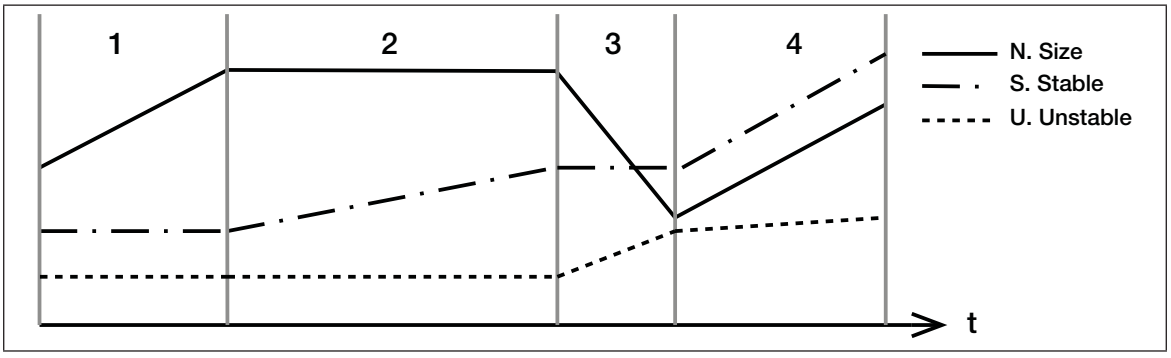

Figure 3.5 Evolution of properties over time of one variability zone in four phases.

but realistic example of variability zone evolution illustrated in Figure 3.5. At step 1, new variations are being added, and thus N increases. However, S remains low, which suggest that the stability point is not coupled to a lot of other elements (i.e., low degree of usage). At

step 2, an increasing number of elements refer to the stability point directly. One potentially interesting observation is the fact that this increase in usage does not lead to an increase in unstable coupling in this case, which can be a good sign initially. At step 3, although the number of variations decreases, an increasing number of clients know their concrete implementations and should be verified to check for opportunities to reduce U (e.g., introducing a Factory could help decrease it if the clients are instantiating the implementations themselves). Finally, at step 4, several new variations are being added, and more couplings are toward the stability point interface than toward the variability zone (i.e., S is higher than U).

Views in Figures 3.4 and 3.5 complement each other. The first one (Figure 3.4) is a more detailed view, and depicts both the evolving software design, its structural changes over time (e.g., modules or links added or removed), and variability zones (e.g., added or changed). The second view (Figure 3.5) focuses on showing the evolution of one or several variability zones, in terms of its properties.

The diagram presented in Figure 3.5 is useful to track evolving variability in a software design because it makes it easier (1) to find which variability zones are evolving in a software history, (2) to locate the time periods where properties increase or decrease. For example, for a specific stability point, it is possible to see whether it is being used by many clients (a greater value of S implies a popular abstraction), if it has been extended over time (an increase of N signifies a useful variability zone), or if it has only one implementation (a value of N=1 implies a design of questionable usefulness).

The concepts we presented in this section inspired analytical tasks we asked participants to perform using our prototype.

3.4 Visualization Tool

IHVis allows users to examine source code repositories of Java projects to find and analyze variability zones. To facilitate the exploration of variability in evolving software, our prototype include features and interaction techniques to visualize data using multiple views. In this

section, we will first explain how data is mapped to software designs (modeled as networks), followed by an overview of features, and examples of practical uses of the tool.

3.4.1 Modeling source code histories as dynamic graphs

IHVis reads the change history of source code in repositories (such as Subversion), that include a list of transactions in which files are modified by developers. A network representation can then be built for each transaction (numbered with a unique revision number) by parsing changed files as shown in Figure 3.6.

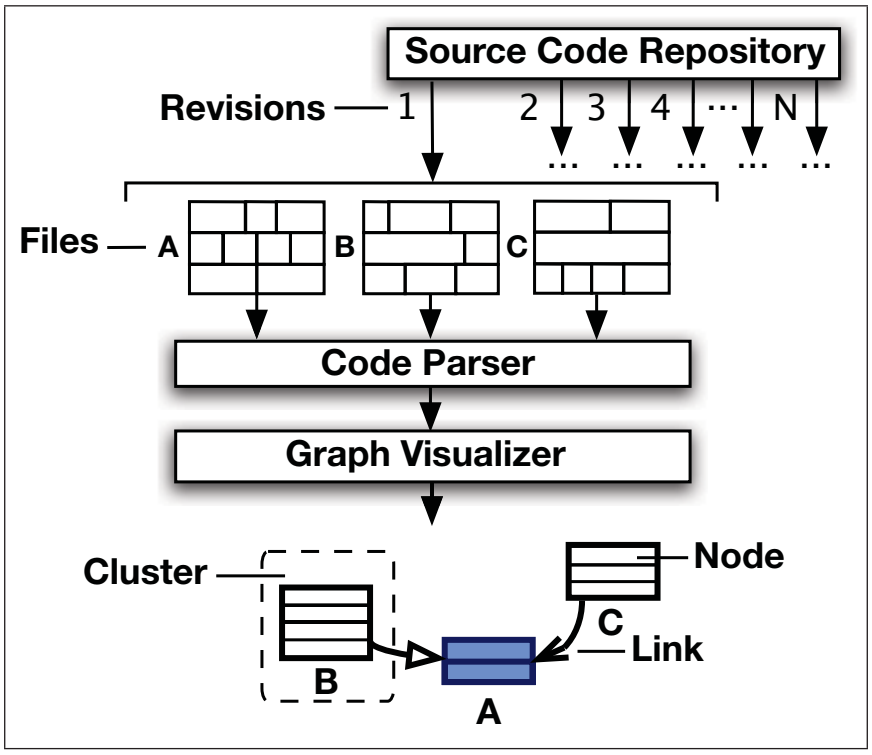

Figure 3.6 Ilustration of the data collection and visualization processes in IHVis.

The parsing process allows our prototype to detect changes at different levels of abstraction. Coarse-grained changes (such as classes or interfaces added, modified or removed), and lower level changes (such as modifications of attributes, method signatures or code implementations) are thus collected for each revision. A network representation can naturally represent this data

model. In this case, nodes include classes (and their methods), interfaces and abstract classes, while couplings, inheritance and interface extensions are examples of edges. Furthermore, following our definition of variability zones (see section 3.3), the implementations and extensions of a stability point (a Java interface), form network clusters.

The result of the data collection process shown in Figure 3.6 is a model, in which software networks are transformed over several time steps, and it can be further visualized in our system.

3.4.2 Visualization prototype

Common ways to visualize dynamic networks include small multiples, difference maps (Archambault *et al.*, 2011b), and animation. Small multiples essentially juxtapose static representations of the graph at different time slices, while a static difference map also highlight differences between two time slices. An animation smoothly interpolates changes between two time slices. In previous work, researchers in visualization have found benefits of using static representations (Archambault *et al.*, 2011a; Farrugia et Quigley, 2011) and animation (Zaman *et al.*, 2011). Because of the possibly complex trade-offs between the techniques, we let the user decide which approach he/she prefers to use in our prototype.

The workspace of IHVis is composed of multiple views, each focusing on a different perspective of evolving stability in designs, as shown in Figure 3.7. A video demonstrating the tool is also available online [3].

The user can interact with two views that are displayed at the same time : a zoomable network view (Figure 3.7A,B) and also a time-line view, showing either the evolution of variability zones' properties using line charts (Figure 3.7C) or the evolution of stability and project metrics using bar charts (Figure 3.7D).

To simplify visualizing complex projects with lots of classes, our tool supports panning and zooming in all views as well as highlighting and filtering of classes. Classes can be filtered by name or object type, and by making rectangular or manual selections. Users can also browse

3. http://ref.rufiange.com/ihvis2013

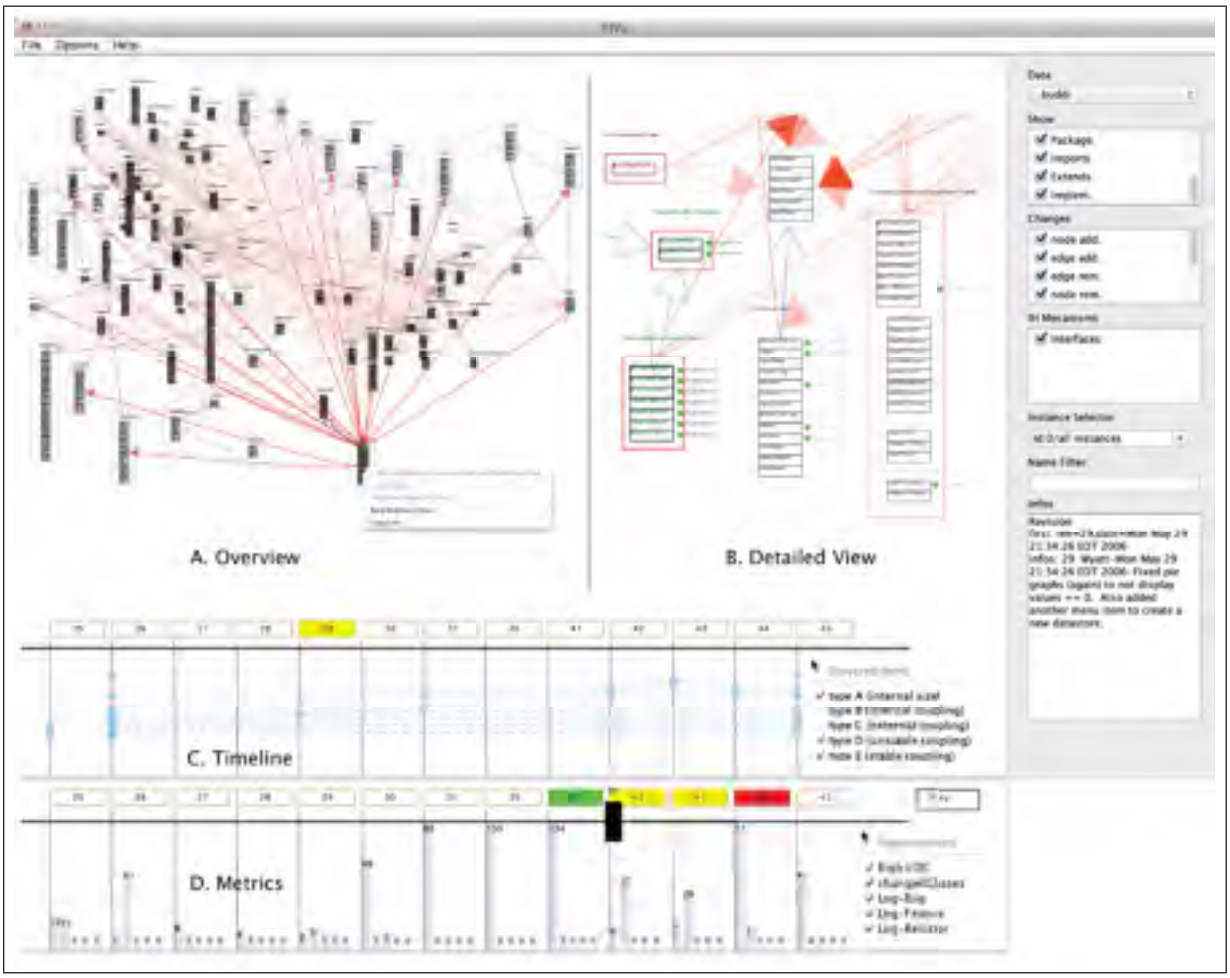

Figure 3.7 IHVis displays a network view (e.g., A or B), along with a time-line view (e.g., C or D) in a single workspace. **A**. Higher level view of a software design showing general tendencies. The user can zoom and pan around the view and see relationships between elements. **B**. Detailed view showing different couplings (inheritance in green, implementations in blue and others in red), addition (in green) and removals (in red) of elements and other changes between revisions 200 and 230 of Buddi. Illustrated here are many changes in the signatures of methods, and some implementations as well (changes to modifiers are also supported but not shown here). **C**. Visualization of the evolution of the variability zone properties (internal size, stable and unstable couplings). **D**. Bar charts showing metrics of revisions (e.g., LOC, classes modified, keywords match in commit messages).

the list of stability points that were found and focus on only one of them as desired. If a filter is applied, only the elements in focus and their related elements (neighbors) are visualized. A blank workspace is created when a user imports a new dataset from a source code repository. In this case, nodes positions are computed using a force-directed layout (Fruchterman et

Reingold, 1991), for each time step of the dynamic graph. However, graphical elements can be freely moved around by drag and drop (workspace sessions can also be saved and restored later on). To help preserve familiarity with UML, known to most software engineers, children elements (such as internal classes or methods of classes) are shown using nested enclosures and indentation.

The network view can be used to show the software design at a specific moment in time or to compare two time steps. If a single time step is selected, the data is visualized using a small multiple. If several time steps are selected, their differences are highlighted, and animated transitions are used to show the changes. The visualization of these changes may help reveal observations about a design (such as the removal or addition of software elements and couplings, the nature of the changes and the affected variability zones). Also, in a zoomed-in view, the user can see the details of the methods of the visible classes and interfaces including access-control information (visibility).

IHVis shows the software elements or nodes (interfaces, classes) and the types of relationships (extends, implements or imports) using a UML-like notation shown in Figure 3.8. Variability zones show as blue dashed areas associated with a respective stability point. The dashed lines are easier to identify in complex visualizations. Since standard UML contains dashed lines

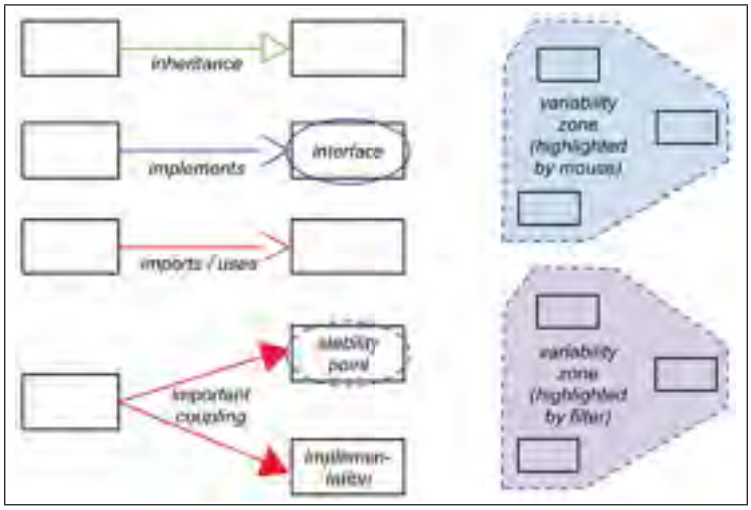

Figure 3.8 Notation within IHVis' structural view.

(e.g., to indicate interface implementation), they detract from the pre-attentive processing benefit for identifying variability zones. This is the main reason for a notation that is different from UML. Also, when a node (or a cluster of nodes) is hovered, its neighbors are highlighted. This is useful to see more clearly the nodes responsible for unstabilities in a variability zone.

Although the network-based visualizations are helpful to show evolving changes in software designs, they do not support the user in finding potentially important time steps in terms of stability. To facilitate this higher level exploration of change histories, IHVis implements time-line views (shown at the bottom of the workspace). The selection of time steps is done by interacting with the time-line view. Users can switch to a different revision shown in the time-line view by clicking on it, and a range of revisions can be selected by holding shift and choosing the starting and ending time steps. Furthermore, the time-line is browsable, and so the user can move forward (or backward) in time by click-dragging the mouse. The user may control the speed at which the animated transitions are shown by scrolling in time manually using a scrubbing technique or automatically by pressing the play/pause button.

This interactive and filterable time-line view can display one of two possible visualizations at the same time. The time-line view can first display line charts (Figure 3.7C), showing the evolution of variability zones' properties. In this case, a user can track fluctuations in terms of size of variability zones (N) or its stable and unstable couplings (S and U, respectively). A second possible visualization for the time-line view is bar charts, that display the evolution of project metrics (Figure 3.7D). Bar charts thus displays other information about the changing designs over time, such as Lines of Code (LOC), number of changed classes, and number of occurrences of keywords in commit messages, for specific revisions. These views support the user in browsing (possibly large) change histories, looking for stability evolution patterns.

3.4.3 Stability Points in the Wild

This section presents results of noteworthy observations made by the authors, using IHVis on different open-source projects mentioned in Table 3.1. JabRef (http://jabref.sf.net) is a

Tableau 3.1 Projects that were explored using our approach. JHotDraw and Buddi were also part of the user study.

Project	# Rev.	# Files	Size	Dates
JHotDraw	781	484	3.1 MB	2000-2012
Buddi	1234	253	1.6 MB	2006-2013
JabRef	3746	581	5.8 MB	2003-2013
Violet	284	290	1.9 MB	2010-2013

bibliography reference manager and BibTEXeditor, and Violet (http://violet.sf.net) is an application for drawing UML diagrams, whose architecture is documented in (Chansler *et al.*, 2011). JHotDraw (http://www.jhotdraw.org) is a project that uses a large number of design patterns and, as such, can be seen as both a graph-drawing framework and application. Buddi (http://buddi.digitalcave.ca), on the other hand, is a less complex project to manage personal finances and budget of households. The focus here is on the early iterations of the software projects, with the assumption that iterative designs tend to be less stable at early phases (Jacobson *et al.*, 1999).

3.4.3.1 Buddi

Using our tool, we discovered that there were several interfaces with a small number of extensions (e.g., ModelFactoryImpl, SourceImpl, TransactionImpl) across the Buddi change history, as shown in Figure 3.9.

At first, we could conclude that it might be a case of adding unnecessary dimensions of extensibility or over-designing [4]. However, upon further investigation in the source code and using the time-line view, it appears that some of these implementation classes are in fact models in

4. http://www.martinfowler.com/ieeeSoftware/continuousDesign.pdf

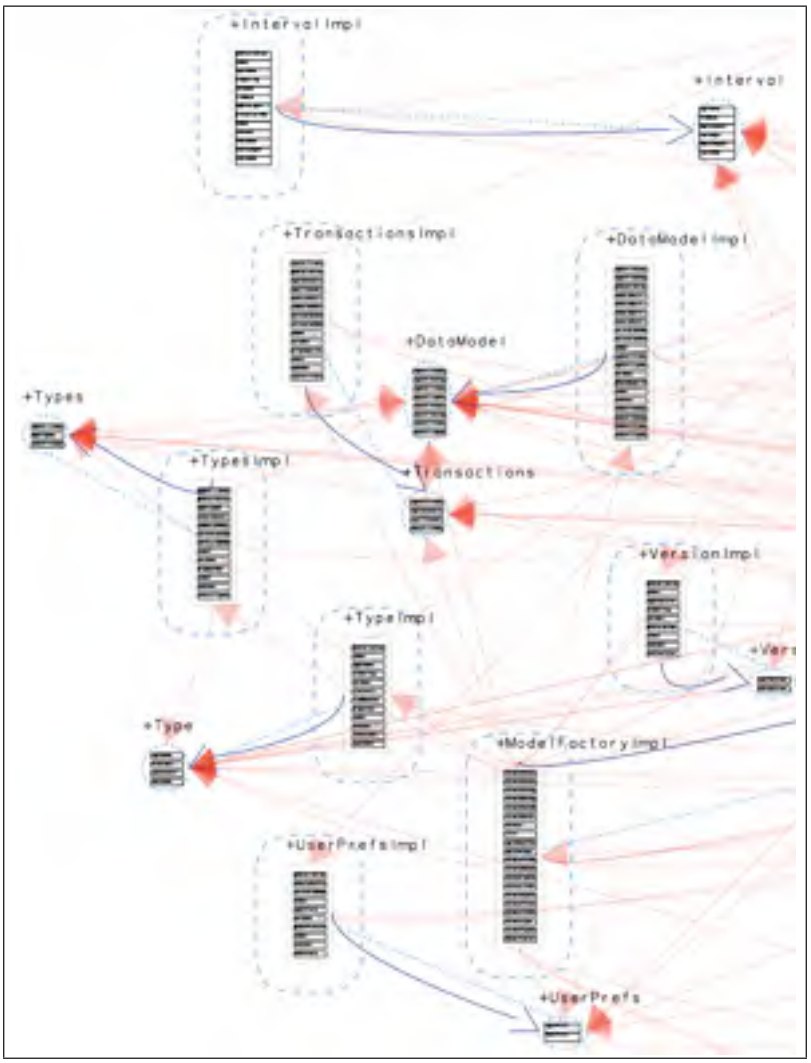

Figure 3.9 In Buddi, there are several small-sized variability zones, because they are backed by the Eclipse Modeling Framework.

a data-model framework, as they are extending the Eclipse Modeling Framework classes (e.g., org.eclipse.emf.ecore.impl.EFactoryImpl and org.eclipse.emf.ecore.impl.EObjectImpl).

At some point after revision 225, the usage of several stability points (e.g., Transaction, Category, PrefsPackage) does not change for a long period of time, suggesting the stabilization (or maturity) of a part of the design (see Figure 3.10A). In another view, the AutoComplete variability zone was modified around revisions 108 and 112 in order to (citing the revision comment) “rearrange internal text component classes and move Hint code from the container

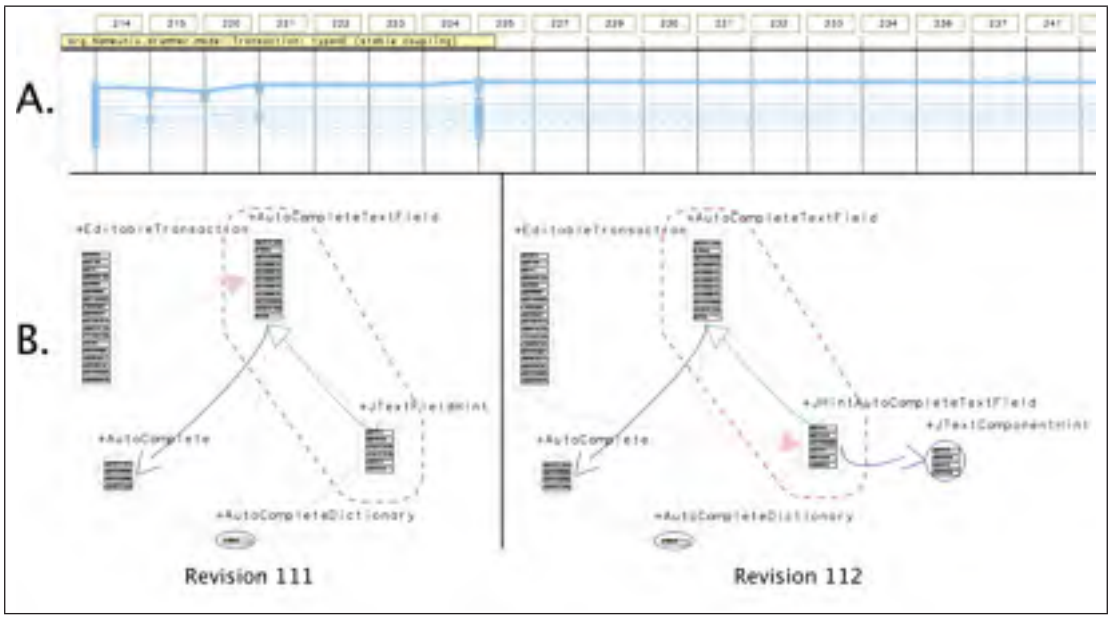

Figure 3.10 Illustration of IHVis with the Buddi project. **A.** The stable coupling of the Transaction stability point, among others, remains constant near revision 225. **B.** Introduction of a new class to replace an old one in the AutoComplete stability point at revisions 111 (on the left) and 112 (on the right).

to the components". In fact, Figure 3.10B shows this small refactoring visually, as the developers introduced a new class (JHintAutoCompleteTextField) to handle the auto-completion of text fields that replaces the older class JTextFieldHint.

Another example of a design improvement is with the BuddiReportPlugin (revisions 261-265, Figure 3.11) in which the developers worked on the "dynamic loading of reports, . . .custom reports options . . .within the new plugin architecture". Before the introduction of that stability point, the code to generate reports was located inside ReportFrameLayout. However, as a new kind of report is added (IncomeExpenseReportByDescription), the code to actually perform the function (e.g., getTreeCell) is moved to nested classes. A possible consequence of this refactoring is that external clients have less access to nested classes.

3.4.3.2 JHotDraw

In the JHotDraw project, a Painter describes different strategies to draw a scene, either by simply looping through all objects or using some caching mechanism to only redraw drawing

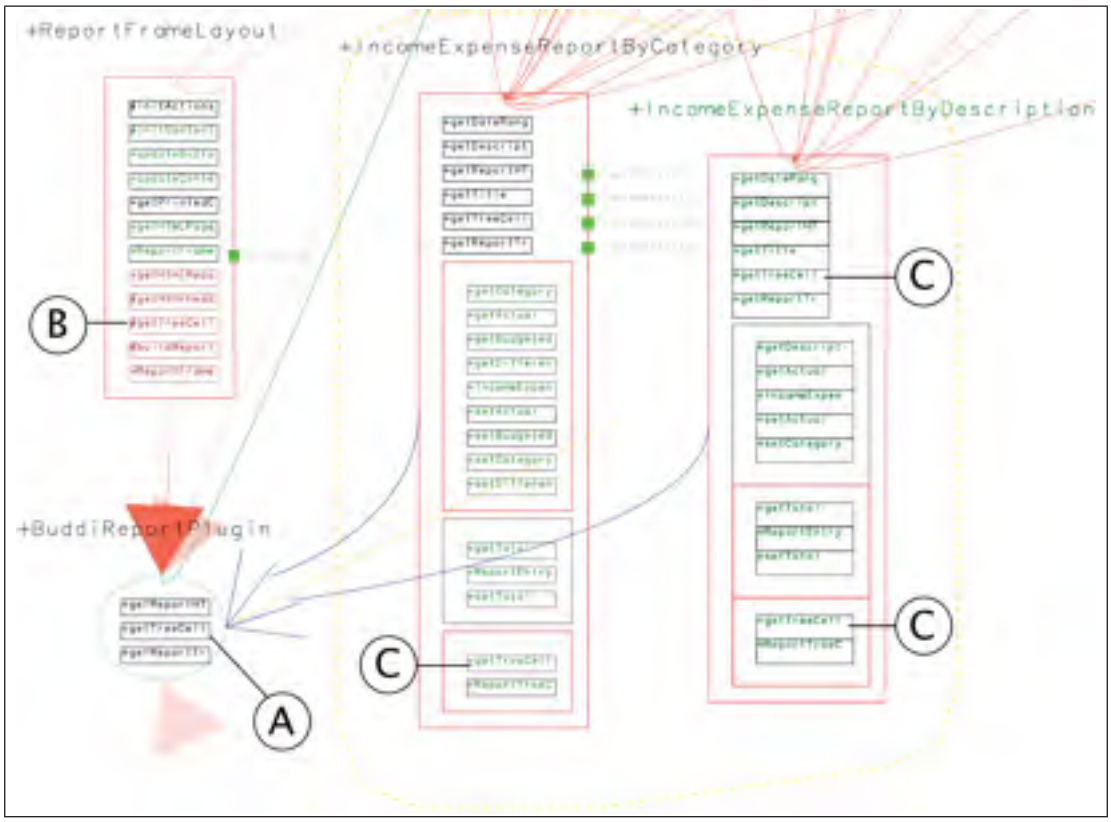

Figure 3.11 A new stability point, BuddiReportPlugin, is added to the Buddi project at revision 262. The changes between revisions 262 and 266 are shown using an animation with IHVis. In this case, implementations (methods) defined by the interface **(A)** are moved from the ReportFrameLayout class **(B)** into new classes **(C)**, to support a new plugin architecture.

areas if required. We can see that new update algorithms, such as ways to draw scaled objects (e.g., ZoomUpdateStrategy), were added most notably between revisions 40 and 45 (see Figure 3.12A). Since drawing views are directly coupled to the concrete update strategies, the unstable coupling fluctuates accordingly (see Figure 3.12B). In addition, a few application classes (e.g., DrawApplet) also contribute to the unstable coupling. Of course, the introduction of a factory could have reduced the degree of unstable couplings and prevent possible future unwanted relationships, although the complexity seems fairly manageable at this point in time.

Similarly, a Layouter's implementor defines how to position objects in space (e.g., SimpleLayouter, HTMLLayouter). At revision 45 the developers refactor and migrate code from

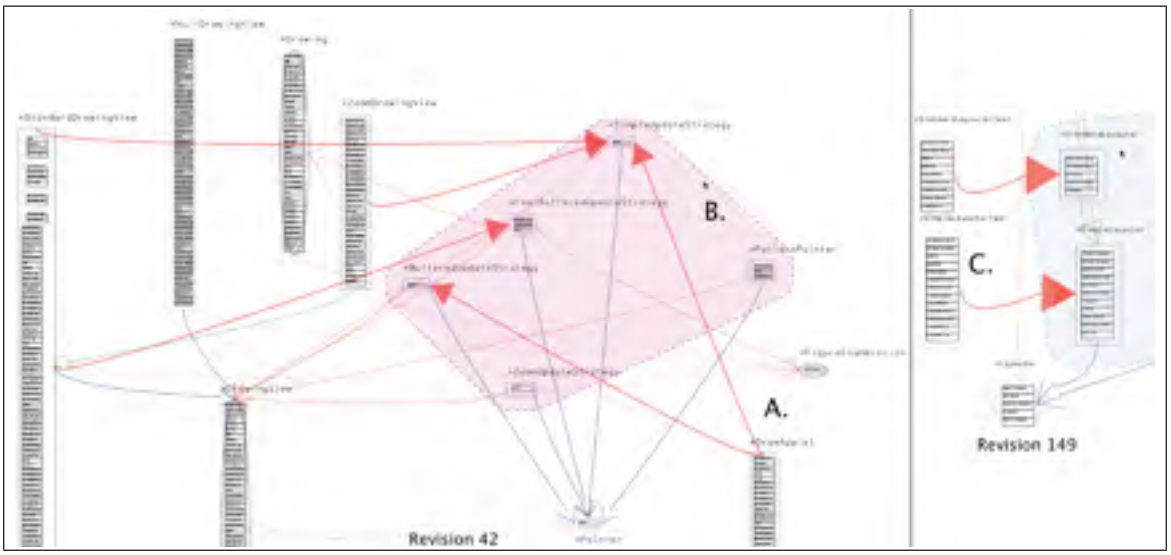

Figure 3.12 Illustration of IHVis in the JHotDraw project. **A.** New update strategies are added at revision 42 (e.g., ZoomUpdateStrategy). **B.** Several drawing views classes are coupled to implementations in the variability zone. For example, ZoomDrawingView was just added, and thus also increases the unstable couplings. **C.** At revision 149, test cases (such as SimpleLayouterTest) are added and cause an increase in unstable couplings.

StandardLayouter to SimpleLayouter, and reverse the "extends" relationship (see Figure 3.13).

Also, at revision 149, there is a significant increase in unstable couplings, which is essentially caused by the addition of test cases (e.g., SimpleLayouterTest) (see Figure 3.12C). Interestingly, further observations in the source code suggests that the designer seems to use a top-down methodology in this case. In addition to the use of test cases, the code for HTMLLayouter, for example, remains defined only at a higher level and is left not fully implemented for some period of time.

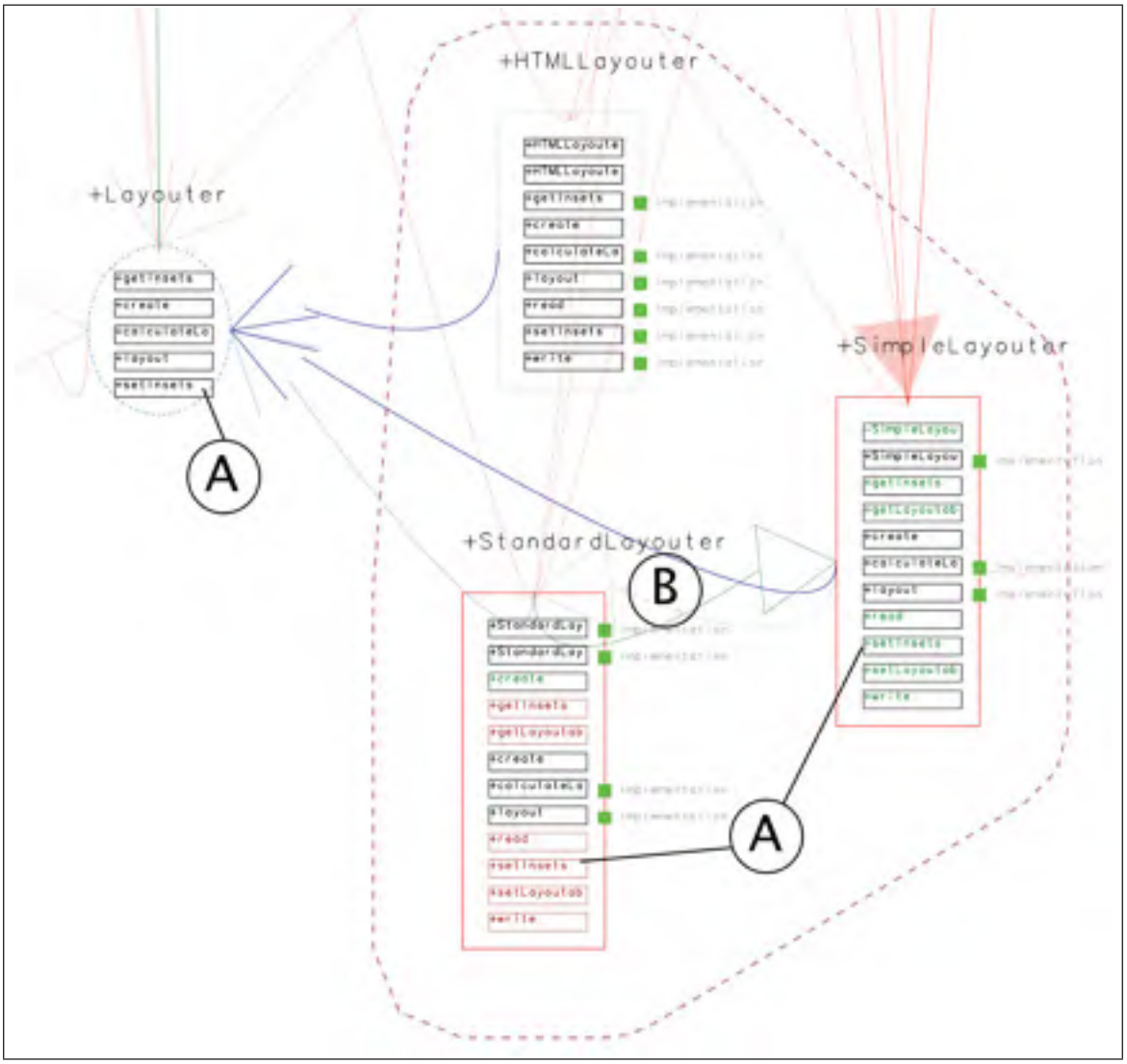

Figure 3.13 Refactorings in the JHotDraw project. A variability zone evolves and two implementations are also switching their inheritance relationships. **(A)** Some methods defined in the stability point (such as setInsets) are redefined and several methods from StandardLayouter are migrated to SimpleLayouter. **(B)** SimpleLayouter was extending StandardLayouter in the past (the inheritance is shown in light green) ; it is now the opposite, i.e., SimpleLayouter is the default base class that StandardLayouter extends (this new refactored inheritance is shown in dark green).

3.4.3.3 JabRef

The PrefsTab interface in the JabRef project has an increasing number of concrete implementors (starting from revisions 31) which define types of Tab (e.g., GeneralTab, ExternalProgramsTab) in the user interface (see Figure 3.14A). A closer look at the time-line view shows that the unstable coupling matches the stable coupling, since the PrefsDialog2 client class creates all the elements inside the variability zone (see Figure 3.14B).

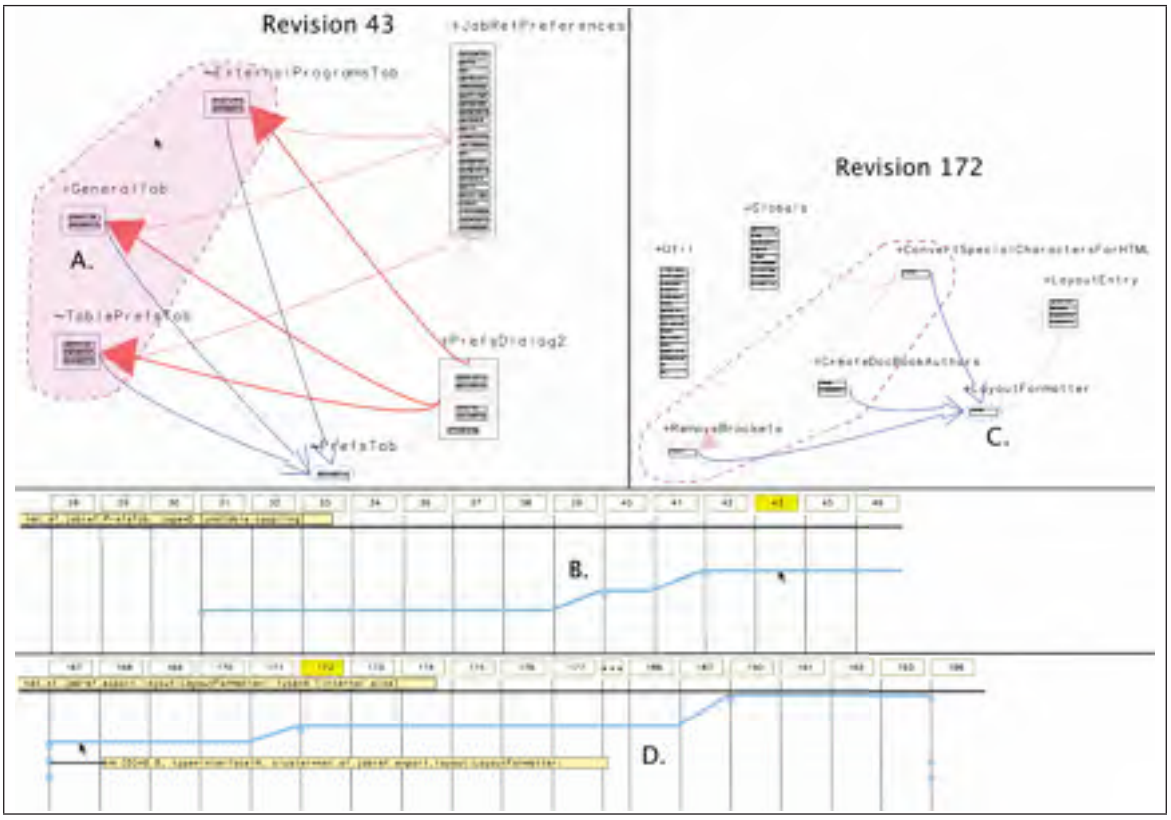

Figure 3.14 Illustration of observations in the JabRef project. At revision 43, several new variations of Tab are implemented **(A)**. Over time, we can see that unstable coupling seems to increase dramatically, but in fact always matches the size of the variability zone (N). Thus, in the figure, the N (size) and U (unstable) lines are superimposed **(B)**. The LayoutFormatter stability point increases in size **(C)**, although there are no apparent unstable couplings in the time line **(D)**.

This is noteworthy because increases in unstable couplings are generally to be avoided. However, they might be acceptable depending on other factors, such as expected minimal reuse and/or short-term design strategy. A visual approach can help investigate various scenarios. In fact, in this case, each tab is a different group of settings that can be edited in a panel interface, and the PrefsTab stability point helps define how to store the settings of each panel. With regards to this stability point, the developer comments in the source code repository log “With this design, it should be very easy to add new tabs later”.

A designer might wonder, based on this level of unstable coupling, if introducing a Factory would have been useful in this case, because the client (PrefsDialog2) would no longer be coupled to the implementations. Indeed, in this hypothetical scenario, the unstable couplings from PrefsDialog2 to the variability zone would go from 3 to 0, and the Factory would manage the panel object creations. However, since the complexity of the constructions of these objects seems rather small, it might be acceptable not to use a factory. Also, looking at the history, we do not see a dramatic increase in U, which could also be an argument for a designer that introducing a factory might not be needed in this context.

Another interesting stability point that we discovered is LayoutFormatter, introduced at revision 143 (not shown in the figure). The time-line view shows increases in variability types, especially between revisions 160 and 190 (see Figure 3.14C). There are no unstable couplings at all, which means that these internal elements are isolated and unknown to other clients, limiting the possible impact if they are modified (see Figure 3.14D). In fact, the LayoutEntry class acts as a reflection-based factory, which means that client classes can create a type of layout by specifying its name only, using the following code : formatter=(LayoutFormatter) Class.forName(formatterName).newInstance().

3.4.3.4 Violet

The UML Editor Violet has also some interesting patterns that can be examined with IHVis. Variations of the IGraph stability point are quickly added at the beginning of the history, as new type of diagrams are implemented. For instance, extensions to AbstractGraph such as Class-

DiagramGraph, SequenceDiagramGraph and ActivityDiagramGraph are being added within revisions 4-12 (not shown in any figure). Several of these extensions are then disconnected from the diagram.AbstractGraph class and now extend diagram.abstracts.AbstractClass. This transfer of variability (and partial relocation) is also depicted visually as crossing lines in the time-line view, as extensions are moved from one variability zone to another one (at revisions 46-60 in Figure 3.15).

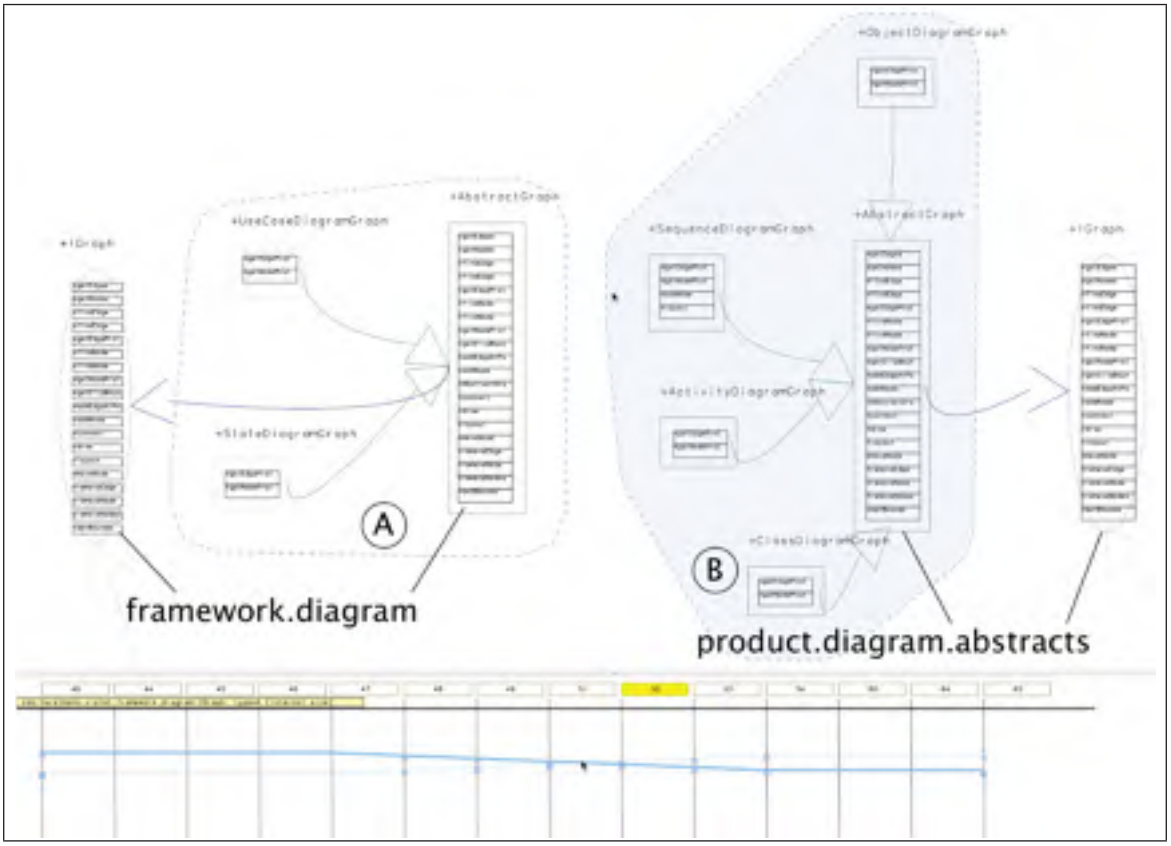

Figure 3.15 Illustration of a case of variability transfer occurring in the Violet project between the revisions 47 and 53. The time-line view shows that, over time, several elements in the variability zone on the left **(A)** are migrated in the variability zone on the right **(B)**. At the same time, it also shows a package restructuration, as the elements on the left area are located in the "framework.diagram" package, while the elements on the right area are located in the "product.diagram.abstracts" package.

As new tools are implemented, new variations of the IEditorPartBehavior interface are added between revisions 129 and 145. Also, in that same period of time, a few classes are coupled to

the interface, and the unstable couplings follow a similar tendency as the increase in variation types, as shown in Figure 3.16A. This phenomenon is caused by classes such as the Workspace

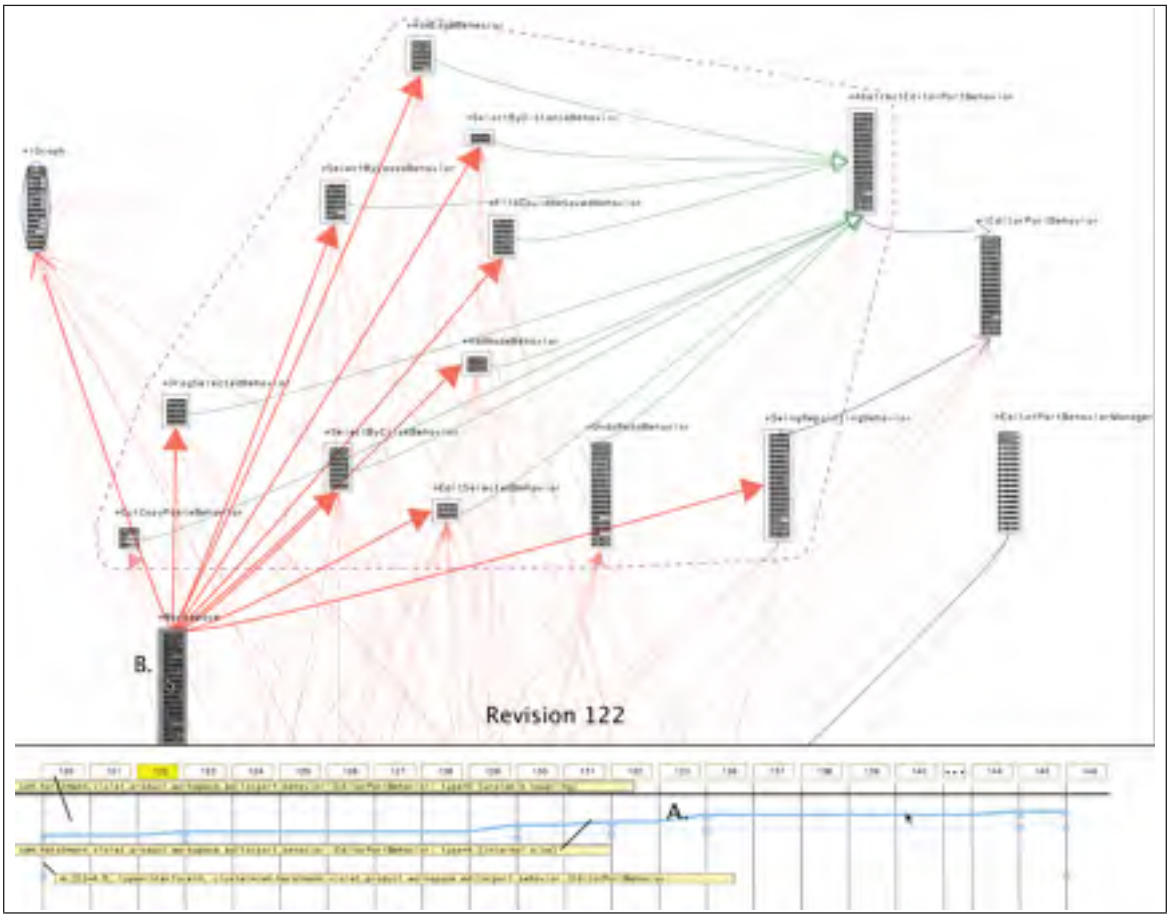

Figure 3.16 Illustration of observations in the IGraph variability zone of the Violet project. Several new variations of the IEditorPartBehavior stability point are added over time between revisions 129 and 145 (causing N to increase). The unstable couplings follow the same tendency as N, as shown in the time-line view **(A)**, in part because of the Workspace class **(B)**.

class (3.16B), which is usually coupled to all tools that are supported over time. This can also be spotted visually by the symmetry of the N and U lines in the figure. A few other classes contribute to the stable couplings (e.g., IEditorPartBehaviorManager).

The developers of Violet also defined the concept of a theme, which is implemented in varying types (e.g., EclipseTheme, DarkTheme, VistaBlueTheme), as shown in the Figure 3.17A. We

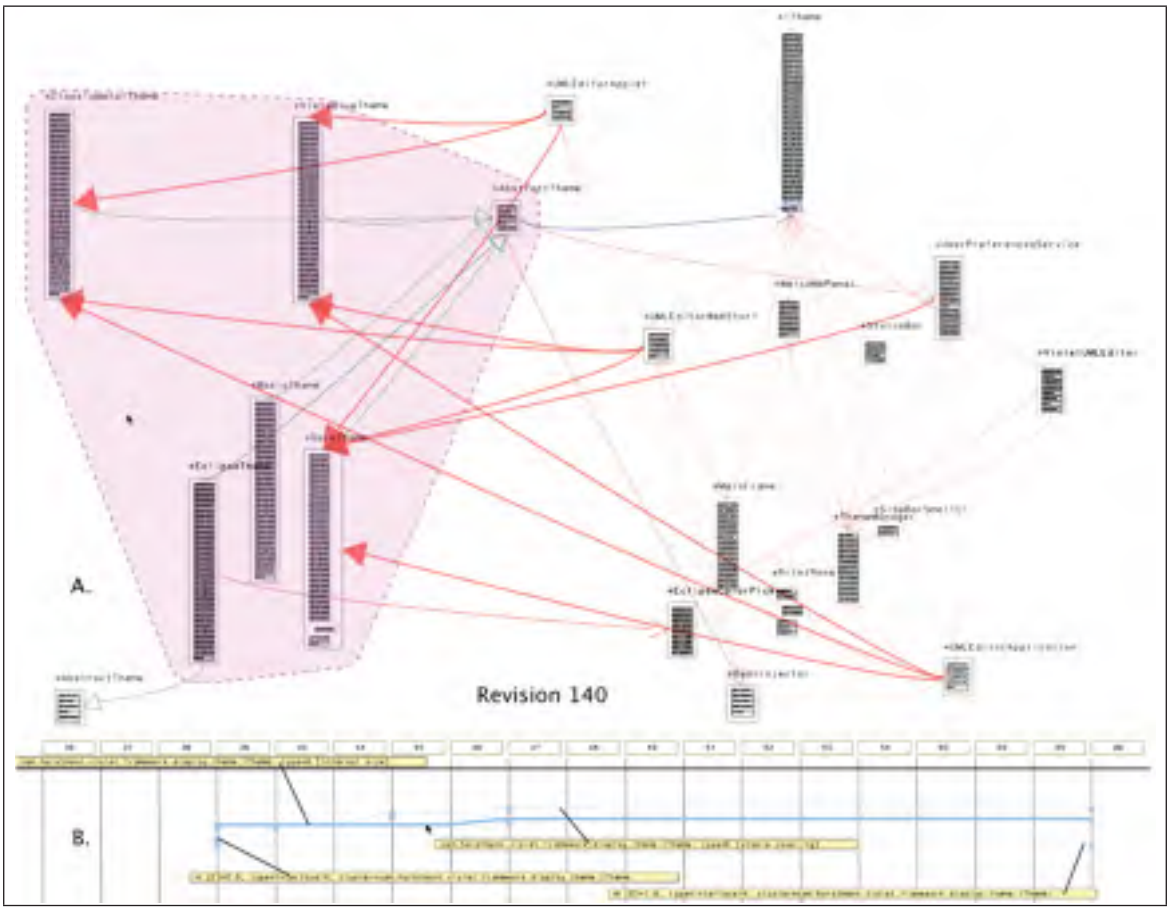

Figure 3.17 Illustration of observations made regarding the ITheme stability point in the Violet project. At revision 140 (after the time line in B) a number of GUI classes are coupled to the implementation Theme classes **(A)**. The unstable couplings (dashed line at the bottom, marked as "D type") remain below the stable ones. The U and N values remained constant in later revisions not shown in part **(B)**.

can observe that the N property of the stability point corresponding to this notion (Theme) increases over time, since the features were added incrementally. Looking at the Figure 3.17B, we can also see that the unstable coupling fluctuates but remains below the level of stable couplings between the revisions 39-66. Upon further investigation, we discovered that the same GUI classes are directly coupled to the concrete themes (e.g., UMLEditorApplication, UMLEditorApplet), while a larger number of classes are only coupled to the associated interface (e.g., VioletUMLEditor, PrintPanel).

We presented in this section our analysis of variability zones discovered in open-source software designs using IHVis. In the next section, we discuss a user study in which we asked participants to find observations about stability in evolving software designs.

3.5 User study

This section presents an investigation of the use of IHVis with human participants. The objective of our study was to see how the participants were able to use our visual tool to find structures in software histories, with a focus on variability zones.

We thus asked participants to perform pre-determined tasks (with expected outcomes) using the IHVis tool. All the tasks were evaluated by having participants answer multiple-choice questions concerning *a priori* observations identified by the authors. A multiple-choice question could have one or more correct answers, as well as wrong answers and an "I don't know" answer. We allowed participants to indicate their own observations, in addition to the ones that were expected. Previous work has referred to recording observations as *insights*, where an insight is an "individual observation about the data by a participant" (Saraiya *et al.*, 2005a) that is noteworthy in the context of the task at hand.

We first did a pilot study with three participants to compare IHVis with a traditional approach to analyze variability zones (Task 3b). The classical approach involved Eclipse and a UML reverse-engineering tool called ObjectAid UML Explorer (http://www.objectaid.com). We focused on this analysis task only, since it is the more complex, and other tasks cannot be performed with the traditional approach. In this pilot test, participants had to identify and document the hierarchies of stability points and construct diagrams at a number of revisions. They were then asked to count the number of client classes that were coupled to the stability point or its concrete implementations, and report their observations.

The results of this initial study encouraged us to pursue a real experiment with twelve participants knowledgeable in object-oriented software design principles. Our user study involved two of the projects mentioned previously (see Table 3.1), namely JHotDraw and Buddi.

We first presented a general overview of our approach, in which we explained how to perform tasks (using a separate dataset excluded from the full study). Participants could then try our tool until they felt comfortable with it, and we answered any of their questions. Users then performed each of the experimental tasks using IHVis, as we recorded the task durations with a software. We compared their answers with our own, to evaluate the error rates. Participants were encouraged to talk aloud during the experiment.

3.5.1 Tasks

In our study, we asked participants to perform three tasks to evaluate if our tool could support them in finding insights in evolving software. The choice of tasks was motivated based on the paper's main objective to explore evolving variability zones in software designs. To achieve this, we asked participants to follow a basic workflow. A necessary first step is to browse a software history to find potentially interesting periods of time, based on fluctuations in project and stability metrics (task 1) or changes in terms of variability (task 2).

In the first task, participants had to search a software history, by dragging the mouse sideways, and find revisions focusing on particular types of changes (e.g., lines of code, bug fixes, refactorings). Specifically, participants had to find the revisions with the biggest changes, by analyzing bar charts in the metrics view of the tool (as illustrated in Figure 3.7D) and also by filtering commit logs based on keywords (e.g., "bugfix", "refactor"). Examples of possible observations for this task are illustrated in Figure 3.18A.

For instance, significant increases in LOC (lines of code) may indicate that new extensions of stability points were added. In addition, changes made to variability zones to fix bugs should not affect their external clients (i.e., the unstable coupling should not increase at that point). However, refactorings could lead to the creation of new stability points, which could be inspected further in a separate step.

In the second task, participants also had to find possibly important revisions in the change history, but with respect to variability zones using a time-line view (as shown in Figure 3.7C). Using filtering and keyword searches, participants had to report significant fluctuations in terms

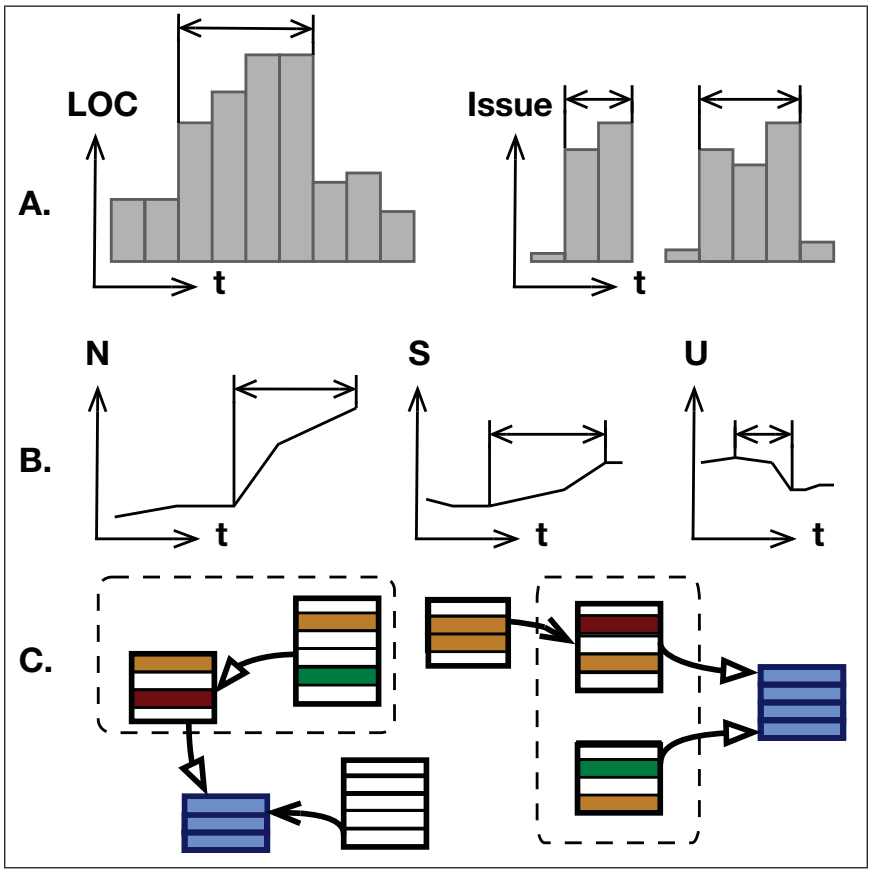

Figure 3.18 Illustration of tasks performed by participants. In tasks 1 and 2, users had to browse a software history and find fluctuations in terms of metrics **(A)** and variability zone properties **(B)**, respectively. In the third task, participants had to analyze stability points and report their findings **(C)**, e.g., cases where changes in variability zones did not affect stable elements (left) or propagated to directly coupled clients (right).

of the properties presented earlier in this paper (note that N, U, S are named respectively A, D, E in the figure). Illustrations of findings for this task are shown in Figure 3.18B. For example, if a variability zone increases suddenly over some time slices (N property), it indicates that new concrete implementations, under the same abstract concept represented by the stability point, were added at some point. Also, increases in stable couplings (S) suggest that more client classes are actually using a stability point over time, while decreases in unstable couplings (U) could have been caused by the addition of a Factory (Gamma *et al.*, 1994) for instance.

Once time ranges possibly containing stability changes are found, a natural third task is to examine in more detail some of these specific cases. We thus asked participants questions

about two instances of stability points, each visualized in an interactive structural view (Figure 3.7A,B). Specifically, we asked participants to inspect two stability points in each open-source project (Buddi and JHotDraw). In the first part of this task (3a), participants had to indicate how the stability point evolved in terms of the N, U, S properties. This part was thus similar to task 2, but they had to filter the data to report observations about this specific case only.

In the second part of the third task (3b), participants had to inspect how the stability point evolved over time (using small multiples and animation) and report their findings. By analyzing and interacting with the visualization, a user could try to understand the evolution pattern of a stability point. For instance, creating stability points can facilitate future changes by integrating a plugin architecture, or changes inside a variability zone could indicate that new algorithms to parse files were added. Furthermore, this analysis may also suggest areas for improvement. For example, a Factory might have been useful to better manage the instabilities of object constructions, although it was not introduced in a software history. We show examples of possible findings in Figure 3.18C. These scenarios were explained to participants, and they had to try to find similar cases in the datasets (or report that they could not find any).

3.5.2 Results

The results of our user study are presented in Tables 3.2 and 3.3 (task durations are shown in minutes). Distributions of the performances and errors of participants are also shown in Figure 3.19. To calculate the percentage of correct answers found and errors, we compared their answers with the expected ones (determined by the researchers). For instance, if the answer was “a, b” and the participant answered “b, c”, it would have resulted in one correct answer, one missed, and another one incorrect (out of two answers).

In terms of task performance, participants were generally able to perform tasks in a few minutes (less than 3 minutes on average). Task 3b not surprisingly took more time than other tasks (approximately 6 minutes on average), because it involved a more complex analysis, such as identifying evolution patterns.

Tableau 3.2 Results of the participants for the Buddi open-source project.

Task	$\overline{x}_{Time}$	# Obs.	$\overline{x}_{\%Found}$	$\overline{x}_{Missed}$	$\overline{x}_{\%Error}$
1	1 :28	4	95.8	4.2	2.1
2	3 :08	5	84.7	15.3	8.3
3a	1 :52	6	69.4	30.6	0.0
3b	6 :05	4	77.8	22.2	25.0

Tableau 3.3 Results of the participants for the JHotDraw open-source project.

Task	$\overline{x}_{Time}$	# Obs.	$\overline{x}_{\%Found}$	$\overline{x}_{Missed}$	$\overline{x}_{\%Error}$
1	1 :28	4	89.6	10.4	10.4
2	2 :54	6	90.3	9.7	1.4
3a	2 :06	5	93.8	6.3	2.1
3b	6 :08	7	89.2	10.8	2.8

Performance of participants in finding correct observations varied more with the Buddi dataset, in comparison with JHotDraw (Figure 3.19B). Participants had more difficulties concurring with the authors about the findings of the Buddi dataset for tasks 3a and 3b, and found ∼75% of the observations, compared to ∼90% for the JHotDraw project. In addition, they made more mistakes (Figure 3.19C), suggesting that Buddi's design was harder to understand (25% of errors vs. 2.8% for JHotDraw for task 3b). The fact that JHotDraw integrated several design patterns possibly resulted in a more decoupled design that participants found easier to grasp.

We also asked participants in our user study to qualitatively evaluate techniques using a five-point rating scale. Participants were fairly confident on average with their analysis (3.9) and they liked the interface of IHVis (4.2), declaring it very useful (4.5) to find observations, even though they never saw or used our tool beforehand.

In the pilot study, we focused on comparing IHVis with a traditional approach involving Eclipse to perform analysis tasks (3b). We were not surprised to find that our approach is faster, since it automates the generation of diagrams and variability zones, and allows visualizing several revisions at once (by animating the changes and highlighting differences).

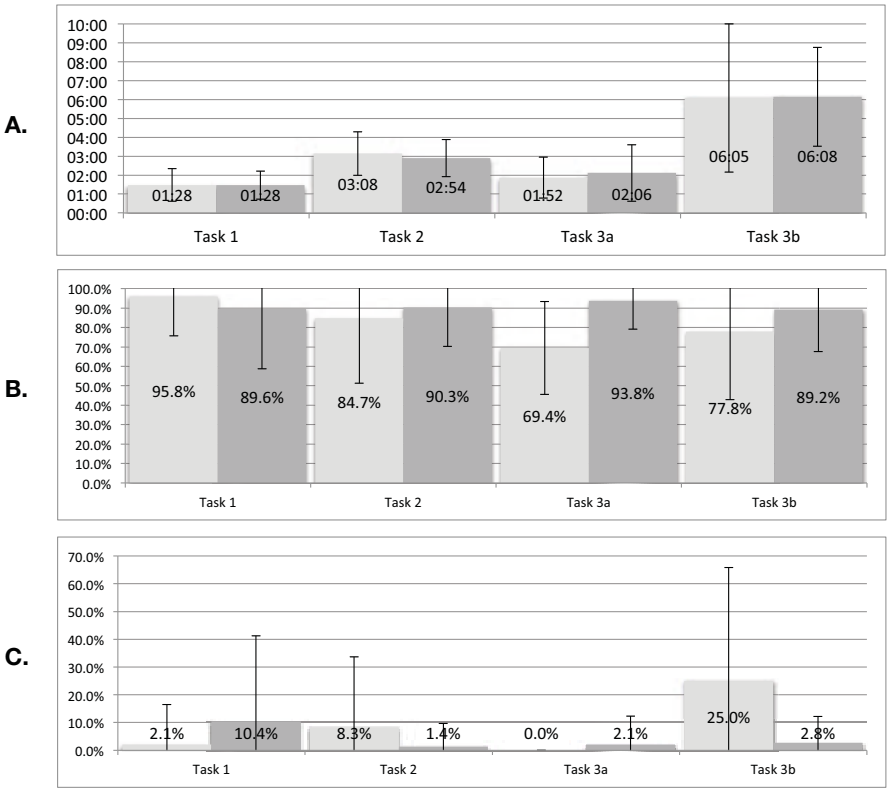

Figure 3.19 Distributions of task durations **(A)**, percentages of observations correctly found **(B)** and errors **(C)** in the user study. Bars are colored to indicate the dataset used, either Buddi (in light gray) or JHotDraw (in dark gray).

The participants' results using Eclipse differed more from those of the authors than when they used IHVis, but participants were nevertheless able to analyze cases using both approaches. However, users reported they lacked information when using the rudimentary approach (they said it was harder to depict changes in code implementation or method signatures). They also had a lower confidence in their results with Eclipse (2.3 vs. 4.0 for our approach), and preferred our tool (they liked it at 4.3 vs. 2.3 for the other one).

Most participants noted that it was easier to understand how the software evolved in time using our approach. In particular, animation, zooming and panning was mentioned as a good means to convey changes quickly. They said it IHVis made it easier to navigate through time and they were able to scan a larger software history. One participant gave up with the Eclipse approach,

arguing that our “approach helped convey evolution information in a much faster and easier manner” and did not want to continue further.

Generally, we believe these results suggest that novice software engineers were able to use our visual tool to find observations about information hiding evolution within a reasonable time frame. Furthermore, participants performed similarly with both datasets, which may indicate that the tool could lead to comparable performances for different datasets. However, there were more variability in results for task 3, probably because participants had different personal experiences in software design.

We also received general suggestions from participants. They reported that the similarity with UML made it easier to understand the visualization. Several users liked and preferred to use the slider to move in time in an animation, although a few others preferred to use the automated play/pause button instead. They suggested we could add other alternative automatic layout algorithms and a more adjustable scale in the bar charts metric view.

3.6 Discussion and Limitations

Using the iceberg metaphor, which is analogous to the open/closed principle, IHVis presents stability points and variability zones respectively, facilitating the inspection of client couplings within a specific frame of reference. Specifically, IHVis helps (1) finding and tracking design structures aimed to support protected variations over time, (2) managing coupling between client classes, the stability points and the implementations, and (3) finding important events about protected variations in an evolving design.

IHVis in its current form has some limitations. Only Java is parsed, and the histories only come from Subversion repositories, although support for GIT is possible by using a data import script (based on the subgit tool). The code must compile if a given revision is to be analyzed (and visualized). The algorithms used to calculate the variability zones (and the respective counts of N, S and U) were kept simple in our tool but are enough to show trends in the data. For instance, IHVis does not capture sub-classes beyond the first hierarchical level by default, although this can be modified in the settings.

Our approach only considers stability points as Java interfaces. This could be generalized to similar concepts in Java and in other languages, such as abstract classes or C++ classes with all virtual methods. Other motives that exercise the open/closed principle but do not involve class hierarchy abstractions could also be stability points. For example, a Facade, Iterator and Proxy provide limited access to elements that could change. These classes need not be defined by an abstraction such as an interface in Java. Instead, a designer could annotate them as stability points. Such annotations could be parsed as per our approach. It might also be necessary to annotate the implementations with respect to the stability point, as they may not be easily deducible. Perhaps in these cases, traditional encapsulation (e.g., with packages and access control) is adequate. The annotation idea could apply to Factories who instantiate implementations, as they are also seen as clients to stability points. Google Guice (Vanbrabant, 2008) makes use of dependency injection to deal with this kind of problem.

Our visual approach makes uses of a node-link view, already familiar to most software engineers knowledgeable in UML, combined with other visualizations and filtering to facilitate the exploration of data. Also, considering the authors' observations presented in section 3.4.3, we believe that our interactive and visual approach, contrary to purely numerical techniques, naturally supports exploratory and qualitative analysis. These features help the designer flexibly discard outliers and focus on interesting cases. As designs are often trade-offs, our approach does not focus on identifying all or automatically classifying correct or incorrect cases. The goal of our method is to help a designer understand how the variability zones evolve and also what changes in the design over time. For instance, if an element will not likely change or is a very stable library, it can be acceptable to have more incoming couplings from clients. Also, as discovered in our observations, unstable couplings caused by test cases can be more tolerable than unstable couplings coming from clients of a framework or API. It puts the burden on test designers to accept the risk of having their code change if the implementations change.

There are also trade-offs to consider when deciding whether to hide implementations to improve a design. Implementations might be relatively unconnected to other elements, thus li-

miting the negative impacts of their changes. Also, stable implementations do not necessarily need to be wrapped under an interface.

The motivation to enforce rules or introduce abstractions will vary depending on the actual project constraints, priorities and objectives, but a "strict" application could mean that an interface should be used whenever the need is apparent to isolate the changes into separate variability zones and limit the propagation to several, possibly increasing, external elements. In this case, a possible guideline would be to use an intermediate class, such as a Factory, that returns an interface to hide the concrete implementations to clients whenever the changes in these implementations start to affect too many external and unrelated clients in other subsystems or packages.

Stability points are possibly a way to validate appropriateness of design patterns (such as GoF (Gamma *et al.*, 1994)) in a project. These patterns make use of stable interfaces to hide possible variations of objects to external clients. If the N, S and U counts with respect to a design pattern's stability point(s) have anomalies in a project's history (e.g., the Observer pattern is introduced, but N and S remain small), that could indicate a bad design choice. Conversely, if N and S continue to grow over time in a project, it could mean that the pattern is fulfilling a need within the design space.

3.6.1 Threats to validity

The user study that we performed aimed at evaluating the usefulness of our novel approach to explore variability zones in software. We could not find an alternative approach to quantitatively compare our tool against. However, we still compared a subset of IHVis with a classical method (using Eclipse with a UML plugin) in a pilot study. Since our tool clearly outperformed the traditional approach, we were encouraged to perform a full user study.

In our real experiment, we then asked participants to use our tool to analyze software designs, following a within-subject design (they thus had to perform all tasks in sequence). To reduce the confounding effects due to task ordering, a Latin-square design determined the order of

tasks and a random number generator determined the order of the open-source projects among participants.

We measured the task durations and error rates using a software program. However, the correctness measurements were based on the participants' ability to find the same insights as the authors, which were not verified by the software developers. Also, in computing the depth of variability zones (e.g., the N property), the number of intermediary levels was fixed to 1. This choice was made so the user could focus on smaller variability zones and discover more specific cases, rather than more abstract ones, which could have been harder to interpret.

The qualitative results of the user study show that participants found IHVis to be useful in solving tasks related to the research goals. Participants in the user study had to answer questions among a list of pre-determined observations (to facilitate evaluation) but were also allowed to state other observations. The feedback we received from participants in our pilot and real studies, and error rates suggest that our tasks were fairly complex but realistic.

The high percentage of insights found in Tables 3.2 and 3.3 show that many of the participants identified most of the important observations in the presented datasets, which suggest that the tool is useful. However, another possible experimental methodology (North, 2006) is to let users find and list all the insights they discovered on their own. A possible issue in this case is that the participant might not be sure of what to look at, especially if advanced knowledge is required in a domain, as in our case.

We also demonstrated how our tool can be used in practice in case studies. As with any case study, this evaluation remain qualitative, and essentially show that our novel approach is working, and can lead to discover insights about software designs.

With respect to external validity, users that participated in our study only analyzed a limited history of two open-source Java projects in Subversion repositories. Furthermore, the case studies that we presented also only covered a small number of projects.

Mean times are for participants in our study, who were volunteers within an academic community and may not represent a general software engineering community. In an effort to recruit a representative sample of human subjects, we verified that participants had completed at least one course in object-oriented software design or had equivalent practical experience. Although results are encouraging, further studies are required to investigate wether they can be generalized to different applications.

3.7 Related Work

The software visualization community aims at finding valuable observations concerning a software and convey how it is constructed and how it is evolving over time. Previous work has focused on visualizing different aspects of software and its evolution (see surveys (Diehl, 2007; Caserta et Zendra, 2011; Khan *et al.*, 2012)). There are also non-visual approaches to studying software design structures and their evolution.

3.7.1 Visualizing static aspects of software

Different approaches have been used in the literature to explore a single version of software, and some of them can be used to compare two versions of a software by juxtaposing static representations. Lattix (Sangal *et al.*, 2005) makes use of matrices to reduce the problem of occlusion for software with a lot of dependencies, but does not support the browsing of software histories. Abdeen *et al.* (2010) proposed a visualization in a matrix-inspired form to see more detail about package relationships, particularly raw size of a package, the incoming and outgoing coupling, as well as cohesion. They consider in their visualizations the visibility of classes (encapsulation with packages) as well as client coupling to packages. Their approach is limited to packages as a unit of modularity, and only applies to single versions of a software project. Matrices, although scalable, can be complex when the software they represent is complex. To help reduce this issue, TreeMatrix (Rufiange *et al.*, 2012) proposed using them only when requested, in combination with node-link diagrams to facilitate the understanding of higher level software relationships. Holten et Van Wijk (2008) proposed a technique to compare hierarchies of two versions of the same software using a cable-and-plug wiring metaphor.

Classes and hierarchies that are the same in both versions are connected by wires. Hierarchies in both versions are placed opposite each other to make it easier to compare. The notion of a module is limited to the scope of a package.

3.7.2 Visualizing software evolution

Visual approaches have also been explored to study the evolution of software. CodeFlows (Telea et Auber, 2008) allows comparing hierarchies of classes of several file versions to identity changes in code such as merges and splits. Voinea et Telea (2007) implemented a visual data-mining tool to show the changes in source code repositories. Their approach allows different levels of detail in a condensed view to show evolving metrics (lines of code and number of modifications). McNair *et al.* (2007) proposed a technique to filter the evolution data based on change sets, allowing designers to see the changes that were required to fix a bug, for example. However, all of these visual techniques ignore the evolution of coupling and especially in a context of encapsulation.

The Evolution Radar (D'Ambros *et al.*, 2009) focused on visualizing the evolutionary coupling, collected from source code repositories. Wettel et Lanza (2008a) used a 3D building metaphor to represent the activities occurring in a software change history and explored real projects. Langelier *et al.* (2008) used an animated treemap layout to visualize changes in package structure and metrics, and they identified some cases of responsability transfer or growing classes. Other work (Denier et Sahraoui, 2009) has found visual patterns of interest in source code, but cannot be used to show evolving design structures. Although these contributions organize the classes in packages and can convey useful information about structure, they do not support the visualization of arbitrary encapsulation. Hindle *et al.* (2007) used a graph-based layout and colors to show evolution of couplings between packages. None of these methods involved user studies.

3.7.3 Other approaches

Other non-visual approaches have also been used to explore software histories or identify design structures, such as metric-based empirical approaches (Aversano *et al.*, 2007; Alshayeb et

Li, 2005) or change-impact models (German *et al.*, 2009). Contrary to these, visual approaches such as ours have the potential benefit of exploiting the human visual system to facilitate the interpretation of complex data. Furthermore, visual approaches can lead to unexpected discovery through interactive explorations (Ware, 2000) and can be used to convey changes over time.

Concerning the problem of arbitrary encapsulation (which appear at the micro-architectural design level or higher, beyond the traditional package level), some elements in pattern description languages are related to our work. eLePUS (Taibi, 2007) defines several relations that apply to information hiding. The *Encapsulation* relation addresses hiding internal details of a class or group of classes. The *Decouple* relation specifies the notion that two elements are independent of each other such that changes in one do not affect the other. Bayley et Zhu (2010) proposed the *Client depends on Root* (CDR) property, which specifies that if a message is sent from a class that is not explicitly mentioned in a pattern, then the operation must be declared in the root (of a hierarchy) class where that property is specified. This applies to our work because it specifies a kind of encapsulation – clients generally access the elements of a pattern through a stable interface and do not interact with other elements. The concept of a *blackbox framework* (Fayad et Schmidt, 1997) is also related.

3.8 Conclusions and Future Work

Interactive visualizations, such as the one presented in this paper, can naturally support explorations of structures and convey evolution tendencies. Our visual tool (named IHVis) allows exploring software designs that evolve over time. The principal novelty is that it helps designers find and track structures used in the protected variations pattern. By recognizing stability points with our tool, a designer can inspect, understand and possibly correct the application of information hiding.

Analyzing evolution of variability in software designs is still hard to do in current tools, and our main contribution is to provide a method to explore software stability over time. We presented practical use cases of the visual tool to uncover software elements in real software, such as

plugin architecture, variability transfer, and changes in terms of hierarchies of classes. This could help finding good applications of patterns that should be repeated for more projects, or ones that should be avoided. For example, a user might discover that a Factory pattern would have helped isolate external clients from changes in concrete implementations of parsing strategies.

The results of a user study show that the tool is useful and that users were able to confirm insights found by the authors in open-source Java projects (stored in Subversion repositories). Participants were first asked to browse large change histories and report noteworthy revisions in terms of evolution in variability, increased usage of stable interfaces and reduced couplings to concrete implementations of interfaces. Software engineers involved in our study were then able to find interesting changes such as the introduction of interfaces or other software elements, renaming of classes, and increased couplings to elements that might need to be hidden. This further encourage the development of new tools to visually explore and check applications of information hiding concepts and design patterns in evolving software.

In the future, we plan on exploring other information hiding mechanisms in more detail (e.g., intermediary classes), and making incremental improvements to our tool (e.g., supporting different automatic layout algorithms and other types of couplings). Other research directions include the exploration of typical observations in evolving designs, such as the evolution of the Law of Demeter, as well as visual and empirical approaches to study the application and usefulness of design structures and patterns in software.

CONCLUSION GÉNÉRALE

Dans cette thèse, nous avons proposé et validé des nouvelles approches visuelles pour explorer des réseaux et des logiciels, et contribué à résoudre certains problèmes. Ainsi, la nature et la complexité des données rendent plus difficile le choix des méthodes de visualisation appropriées dans certains contextes et elles comportent chacune leurs propres limitations.

Une solution potentielle à ce problème est de rechercher et de concevoir de nouvelles visualisations qui combinent possiblement les avantages de plusieurs approches. Néanmoins, ces combinaisons, bien que prometteuses (Henry *et al.*, 2007), sont encore peu explorées dans la littérature, que ce soit pour visualiser des réseaux statiques, dynamiques ou des logiciels. Or, certaines de ces nouvelles possibilités pourraient améliorer notre habileté à explorer et analyser des données et découvrir des informations intéressantes à propos de systèmes complexes, en comparaison aux approches actuelles.

En termes de contributions, nous avons d'abord documenté, dans les chapitres 1 et 2, une série de possibilités de combinaisons encore inexplorées dans des taxonomies. Par la suite, certaines de ces possibilités ont mené au développement de prototypes. Nous avons trouvé des avantages à plusieurs de ces nouvelles combinaisons et certaines ont aussi facilité la compréhension de données complexes.

En particulier, pour des réseaux comportant des hiérarchies, nous avons conçu TreeMatrix, une combinaison (ou hybride) inspirée en partie de NodeTrix (Henry *et al.*, 2007). Notre approche se différencie de NodeTrix par l'intégration de techniques d'interaction pour mieux distinguer les relations entre les éléments (MatLink (Henry et Fekete, 2007)) et le support des graphes composés, orientés et pondérés. Nos résultats ont montré que la combinaison de matrices d'adjacence et de diagrammes noeuds-liens et les techniques d'interaction a facilité l'interprétation de relations multi-niveaux dans des conceptions de logiciels. Ainsi, pour des questions portant à la fois sur des informations locales (montrées par une matrice, ce qui réduit la complexité visuelle) et sur des relations à haut niveau entre les sous-réseaux de matrices (montrées par des

diagrammes noeuds-liens), plusieurs participants ont obtenu de meilleurs résultats qu'avec une approche alternative que vous avons évaluée (Lattix (Sangal *et al.*, 2005)).

Une autre nouvelle combinaison de visualisations a été explorée pour les graphes dynamiques. Actuellement, un problème est que les changements en termes de connexions entre les noeuds peuvent créer des déplacements plus ou moins importants, ce qui peut rendre plus difficile la traçabilité de noeuds au fil du temps dans des tâches d'analyses. Notre nouvelle approche hybride (DiffAni) a facilité la réalisation de certaines tâches, en comparaison avec des approches existantes dans la littérature, en changeant de représentation visuelle selon le degré de mouvements dans les données. Par exemple, tracer l'évolution du nombre de liens entre des noeuds a été plus performant avec notre nouvelle technique qu'en ayant recours à seulement une animation ou une carte de différences ("Difference Map"). Nous avons aussi expliqué les limitations de notre approche et exploré le cas extrême d'alternances répétées de représentations, qui peut réduire les performances des approches combinées.

Bien qu'un travail antérieur a exploré et validé un hybride de graphe dans des réseaux sociaux (Henry *et al.*, 2008) dans la littérature, il n'existe pas encore d'approches combinées aidant à mieux suivre l'évolution de changements dans des graphes dynamiques pour certaines tâches. Une approche a exploré l'utilisation d'hybrides pour visualiser des graphes dynamiques (Hadlak *et al.*, 2011), mais elle ne peut être utilisée pour varier les représentations à différents moments dans le temps, contrairement à notre approche. Aussi, l'usage de vues multiples coordonnées est une autre solution potentielle, mais elles ne permettent pas d'interchanger interactivement les visualisations dans le but de combiner leurs avantages, occasionnent des changements de contextes significatifs et occupent plus l'espace. Enfin, peu de travaux antérieurs ont validé ces nouvelles combinaisons avec des participants, alors que ces validations externes ont été intégrées à notre méthodologie.

Un autre problème est la difficulté de tracer et d'évaluer des structures complexes dans des réseaux au fur et à mesure qu'elles sont modifiées dans le temps. Pour contribuer à résoudre ce problème, nous avons conçu un troisième prototype (IHVis), décrit dans le troisième chapitre. Ainsi, ce prototype aide à suivre et analyser les changements apportés à des conceptions

de logiciels, afin d'appréhender leurs effets sur la maintenance. Dans la littérature, il existe des visualisations de réseaux permettant de montrer l'état d'une conception à un moment dans le temps ou bien son évolution. Par exemple, des outils peuvent servir à montrer l'évolution du couplage ou de changements (mesures, ajouts ou suppressions d'éléments). Toutefois, il manque des façons d'identifier, d'analyser et de tracer visuellement des structures de conception au fil du temps, en termes de variabilité. Aussi, notre méthode visuelle et interactive facilite la compréhension et la prise de décision dans l'analyse de cas réalistes (et donc complexes), contrairement à une technique qui serait purement numérique.

Notre nouvelle approche a aussi permis de documenter des découvertes intéressantes dans plusieurs projets différents. Ainsi, des cas concrets d'introduction d'interfaces, de restructurations, d'évolution de zones de variabilité et de couplage stable ou instable ont été répertoriés. Les participants à notre étude ont également pu relever plusieurs observations intéressantes au sein de conceptions logicielles. Ces informations ont des significations importantes afin d'aider à tracer l'évolution des conceptions qui évoluent en phases (Jacobson *et al.*, 1999). Par exemple, certains changements vont plus affecter la conception que d'autres à certains moments, et les identifier et les analyser permet de contribuer à éviter des problèmes de *design drift* ou *design erosion* (Perry et Wolf, 1992).

En résumé, nos principales contributions sont la découverte de nouvelles possibilités de combinaisons de visualisations, l'implémentation de prototypes explorant ces idées et leur validation dans des expériences utilisateurs. Les taxonomies que nous avons développées dans cette thèse décrivent des agencements de visualisations de graphes statiques et dynamiques. Certains des cas que nous avons mis en lumière sont toujours inexplorés. Ainsi, nos contributions ouvrent la voie à d'autres hybrides qui pourraient offrir des avantages pour analyser des données au sein de réseaux et de logiciels complexes.

Discussion

Les méthodes de validation utilisées dans cette thèse ont été inspirées de la littérature (voir la revue de littérature à la page 27 pour une discussion des approches d'évaluation). Concernant la validation de nos nouvelles approches, nous avons conçu des expériences appropriées selon les contributions visées. Le premier chapitre (TreeMatrix) visait à faciliter des tâches d'exploration en génie logiciel. Étant donné qu'il existe plusieurs heuristiques (mais pas de règles formelles) guidant la conception de logiciels, nous avons opté pour une évaluation qualitative afin de comparer notre nouvelle approche hybride à un outil commercial reconnu. Pour ce faire, nous avons choisi des tâches typiquement utiles pour les ingénieurs logiciels et reposant sur des principes fondamentaux, tels que l'organisation de modules en couches selon la stabilité.

Dans le deuxième chapitre (DiffAni), l'objectif principal était d'évaluer si la combinaison de techniques de visualisation pouvait être avantageuse dans certains cas. Ainsi, nous avons évalué quantitativement la performance et le taux d'erreurs des participants pour effectuer des tâches d'analyse de graphes décrites dans des taxonomies existantes (voir Lee *et al.* (2006); Ghoniem *et al.* (2005); Ahn *et al.* (2011) pour des exemples). Les participants devaient essayer toutes les techniques afin de comparer les résultats par rapport à notre approche hybride.

Dans ces deux premiers chapitres, nous avons également proposé des taxonomies qui ont été développées itérativement, c'est-à-dire en plusieurs versions successives. Dans le but de construire des taxonomies les plus complètes et utiles possibles, nous avons survolé et référencé tous les travaux pertinents et représentatifs du domaine. Ainsi, une bonne taxonomie doit idéalement permettre de mieux comprendre et distinguer un ensemble de travaux reliés, en les classifiant selon diverses dimensions. Nous avons donc organisé ces classifications logiquement de façon à obtenir un consensus au sein de nos collègues dans un premier temps, puis auprès de la communauté scientifique dans un deuxième temps (via le processus de relecture par les pairs). Dans la littérature, la majorité des taxonomies sont validées selon ce cadre. Une approche de validation complémentaire est l'utilisation de questionnaires (Ahn *et al.*, 2011).

Dans le troisième chapitre (IHVis), afin de démontrer l'utilité de notre approche en pratique, nous avons demandé à des ingénieurs logiciels de tenter de retrouver des observations intéressantes concernant des conceptions logicielles. En comparant leurs découvertes avec ce que nous avions trouvé au préalable, les résultats suggèrent que les participants ont pu bénéficier de notre approche. De plus, nous avons comparé qualitativement notre méthode à une approche alternative potentielle avec un nombre limité de participants, simplement pour nous assurer que notre approche n'était pas moins performante. Enfin, des exemples de cas, retrouvés suite à l'analyse de projets libres concrets, ont été documentés et discutés dans ce chapitre. Dans la littérature, bien que les études de cas soient effectivement répandues en visualisation de logiciels, les interfaces sont rarement évaluées avec des participants.

En termes de contributions internes, nous avons également conçu une plateforme de code dans le but de faciliter le développement de nos prototypes. Celle-ci permet notamment de (1) collecter des données statiques et dynamiques concernant des logiciels, (2) produire des interfaces graphiques interactives avec OpenGL, (3) gérer des fichiers contenant des données de réseaux, (4) contrôler le déroulement d'expériences et collecter les résultats. Cette plateforme n'est pas encore suffisamment complète et documentée pour être rendue publique pour le moment, mais aidera entre-temps à accélérer le développement de nos futurs prototypes.

Perspectives

Suite à la réalisation de nos travaux, de nouvelles pistes de recherche sont envisageables dans le futur. Malgré les percées déjà effectuées pour évaluer certains hybrides dans cette thèse, on connait encore mal les conditions avec lesquelles les hybrides peuvent mieux performer que les autres approches, compte tenu du peu de validation dans la littérature et des résultats contradictoires. Dans ce contexte, il serait intéressant de concevoir un prototype suffisamment flexible pour expérimenter avec plusieurs nouvelles techniques, dont les combinaisons inexplorées décrites dans nos taxonomies. Ces nouvelles visualisations pourraient être catégorisées, puis évaluées selon les tâches, conditions et types de données supportés. Ces approches sont généralement moins familières pour des utilisateurs non experts et donc les performances pourraient s'améliorer au fur et à mesure que les participants expérimentent avec les hybrides.

Un autre axe encore peu exploré concerne la construction de visualisations hybrides. Alors que nous avons trouvé des bénéfices à certaines des combinaisons de visualisations proposées, il manque des travaux pour guider leur création. Par exemple, en ce qui a trait spécifiquement à notre technique hybride pour les graphes dynamiques, nous varions les techniques pour différentes périodes de temps, mais il pourrait être intéressant de combiner plusieurs visualisations dynamiques pour une même période de temps. Ainsi, il serait intéressant d'évaluer, d'une part, la performance de participants à concevoir différents hybrides et, d'autre part, leur capacité à utiliser ces variantes afin de répondre à des questions ou effectuer des tâches. Les retombées potentielles, de même que les résultats obtenus au cours de cette thèse, encouragent la poursuite des efforts dans ces directions.

En outre, ces hybrides pourraient avoir des applications concrètes dans plusieurs domaines, dont en génie logiciel, où les changements varient et peuvent causer plus ou moins de fluctuations en termes de liens et de noeuds au fil du temps (lorsque ces logiciels sont modélisés par des réseaux). Ces changements, orchestrés en phases (Jacobson *et al.*, 1999), pourraient bénéficier de la combinaison de représentations afin de mieux comprendre l'évolution du logiciel.

En génie logiciel, les *points de stabilité* décrits dans cette thèse pourraient en fait se retrouver dans plusieurs autres stratégies de conception, en plus des *interfaces* que nous avons abordées. Les classes intermédiaires (par exemple, Facade dans GoF (Gamma *et al.*, 1994)) et l'encapsulation sont des exemples d'autres mécanismes de masquage d'information prometteurs. Ainsi, la généralisation de ces concepts afin d'explorer l'évolution d'autres types de structures au sein de conceptions logicielles est envisageable.

De plus, il manque des méthodes interactives pour évaluer et comparer des options de conception afin de faciliter la maintenance des logiciels. À cet égard, il pourrait être intéressant d'étudier les liens entre la stabilité des conceptions, les révisions de code et les résolutions de problèmes. Les plateformes de collecte de données et de visualisation conçues dans cette thèse pourront être réutilisées afin d'explorer ces nouvelles perspectives. Pour guider les choix de développement et les évaluations des prototypes, une stratégie pourrait être de collecter des données sur l'usage des fonctionnalités. Enfin, la recherche de phénomènes pouvant apparaître

et évoluer dans le temps (tels que l'application de la loi de Demeter, la propagation de changements en lien avec les exigences, les transformations à multiples niveaux d'abstractions), ainsi que la découverte et l'évaluation de patrons visuels au sein de conceptions en évolution, sont d'autres axes de recherche intéressants.

BIBLIOGRAPHIE

Abdeen, Hani, Stéphane Ducasse, Damien Pollet, et Ilham Alloui. 2010. « Package Fingerprints : A visual summary of package interface usage ». *Information and Software Technology*, vol. 52, n° 12, p. 1312–1330.

Abello, James, Frank van Ham, et Neeraj Krishnan. September/October 2006. « ASK-GraphView : A Large Scale Graph Visualization System ». *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 12, n° 5, p. 669–676.

Ahn, Jae-wook . W., Catherine Plaisant, et Ben Shneiderman. 2011. « A task taxonomy of network evolution analysis ». *University of Maryland, Human-Computer Interaction Lab Tech Report HCIL-2011-09*.

Alam, Sazzadul, Sandro Boccuzzo, Richard Wettel, Philippe Dugerdil, Harald Gall, et Michele Lanza, 2009. *EvoSpaces - Multi-dimensional Navigation Spaces for Software Evolution*, p. 167-192. Springer-Verlag.

Alshayeb, Mohammad et Wei Li. 2005. « An empirical study of system design instability metric and design evolution in an agile software process ». *Journal of Systems and Software*, vol. 74, n° 3, p. 269–274.

Andrienko, Natalia, Gennady Andrienko, et Peter Gatalsky. 2003. « Exploratory spatio-temporal visualization : an analytical review ». *Journal of Visual Languages & Computing*, vol. 14, n° 6, p. 503-541.

Archambault, Daniel. 2009. « Structural differences between two graphs through hierarchies ». In *Proc. Graphics Interface (GI)*. (Toronto, Ont., Canada, Canada 2009), p. 87-94. Canadian Information Processing Society.

Archambault, Daniel, Tamara Munzner, et David Auber. 2009. « TugGraph : Path-Preserving Hierarchies for Browsing Proximity and Paths in Graphs ». In *Proceedings of IEEE Pacific Visualization*. (Beijing, China 2009), p. 113–120.

Archambault, Daniel, Helen C. Purchase, et Bruno Pinaud. 2010. « The Readability of Path-Preserving Clusterings of Graphs ». *Computer Graphics Forum*, vol. 29, n° 3, p. 1173–1182.

Archambault, Daniel, Helen C. Purchase, et Bruno Pinaud. Avril 2011a. « Animation, Small Multiples, and the Effect of Mental Map Preservation in Dynamic Graphs ». *IEEE TVCG*, vol. 17, n° 4, p. 539–552.

Archambault, Daniel, Helen C. Purchase, et Bruno Pinaud. 2011b. Difference map readability for dynamic graphs. *Graph Drawing*, volume 6502 of *LNCS*, p. 50-61. Springer.

Auber, David, Yves Chiricota, Fabien Jourdan, et Guy Melançon. 2003. « Multiscale visualization of small world networks ». In *Proceedings of the Ninth annual IEEE conference on Information visualization*. (Seattle, Washington 2003), p. 75–81. IEEE Computer Society.

Aversano, Lerina, Gerardo Canfora, Luigi Cerulo, Concettina Del Grosso, et Massimiliano Di Penta. 2007. « An empirical study on the evolution of design patterns ». In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. (Dubrovnik, Croatia 2007), p. 385–394. ACM.

Baker, Marla J. et Stephen G. Eick. 1995. « Space-filling Software Visualization ». *Journal of Visual Languages & Computing*, vol. 6, n° 2, p. 119-133.

Balzer, Michael et Oliver Deussen. 2007. « Level-of-Detail Visualization of Clustered Graph Layouts ». In *Asia-Pacific Symposium on Visualisation (APVIS)*. (Sydney, Australia 2007).

Balzer, Michael, Andreas Noack, Oliver Deussen, et Claus Lewerentz. 2004. « Software landscapes : Visualizing the structure of large software systems ». In *Proceedings of the Sixth Joint Eurographics-IEEE TCVG conference on Visualization*. p. 261-266. Eurographics Association.

Bayley, Ian et Hong Zhu. 2010. « Formal specification of the variants and behavioural features of design patterns ». *Journal of Systems and Software*, vol. 83, n° 2, p. 209 - 221.

Beck, Fabian et Stephan Diehl. 2010. « Visual comparison of software architectures ». In *Proceedings of the 5th international symposium on Software visualization*. (New York, NY, USA 2010), p. 183–192. ACM.

Bertault, François et Mirka Miller. 1999. « An Algorithm for Drawing Compound Graphs ». In *Proceedings of Symposium on Graph Drawing*. (Stirín Castle, Czech Republic 1999), p. 197–204.

Bertin, Jacques, 1967. *Sémiologie graphique : Les diagrammes, Les réseaux, Les cartes*. Paris : Éditions Gauthier-Villars.

Bertin, Jacques, 1983. *Semiology of graphics : diagrams, networks, maps*. Madison, Wisconsin : University of Wisconsin Press.

Beyer, Dirk et Ahmed E. Hassan. 2006. « Evolution storyboards : visualization of software structure dynamics ». In *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on*. p. 248-251. IEEE.

Bezerianos, Anastasia, Fanny Chevalier, Pierre Dragicevic, Niklas Elmqvist, et Jean-Daniel Fekete. 2010. « Graphdice : A system for exploring multivariate social networks ». In *Computer Graphics Forum*. p. 863-872. Wiley Online Library.

Brandes, Ulrik et Steven R. Corman. 2003. « Visual unrolling of network evolution and the analysis of dynamic discourse ». *Information Visualization*, vol. 2, n° 1, p. 40–50.

Brandes, Ulrik et Bobo Nick. 2011. « Asymmetric Relations in Longitudinal Social Networks ». *IEEE TVCG*, vol. 17, n° 12, p. 2283–2290.

Bruls, Mark, Kees Huizing, et Jarke J. van Wijk. 2000. « Squarified Treemaps ». In *Proceedings of the joint Eurographics and IEEE TCVG Symposium on Visualization*. (Amsterdam, The Netherlands 2000), p. 33–42.

Buckley, Jim, Tom Mens, Matthias Zenger, Awais Rashid, et Günter Kniesel. 2005. « Towards a taxonomy of software change ». *Journal of Software Maintenance and Evolution : Research and Practice*, vol. 17, n° 5, p. 309-332.

Burch, M., C. Vehlow, F. Beck, S. Diehl, et D. Weiskopf. 2011. « Parallel Edge Splatting for Scalable Dynamic Graph Visualization ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, n° 12, p. 2344-2353.

Burch, Michael et Stephan Diehl. 2008. « TimeRadarTrees : Visualizing Dynamic Compound Digraphs ». *Computer Graphics Forum*, vol. 27, n° 3, p. 823–830.

Burch, Michael, Stephan Diehl, et Peter Weissgerber. 2005. « Visual Data Mining in Software Archives ». In *Proceedings of the 2005 ACM Symposium on Software Visualization*. (New York, NY, USA 2005), p. 37-46. ACM.

Burch, Michael, Michael Fritz, Fabian Beck, et Stephan Diehl. 2010. « TimeSpiderTrees : A Novel Visual Metaphor for Dynamic Compound Digraphs ». In *Proc. Symposium on Visual Languages and Human-Centric Computing*. p. 168–175.

Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, et M. Stal, 1996. *Pattern-Oriented Software Architecture. Volume 1 : A System of Patterns*. New York, NY : John Wiley & Sons, Inc.

Callahan, Jack, Don Hopkins, Mark Weiser, et Ben Shneiderman. 1988. « An empirical comparison of pie vs. linear menus ». In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*. (Washington, DC 1988), p. 95–100.

Card, SK, JD Mackinlay, et B. Shneiderman, 1999. *Readings in information visualization : using vision to think*. Morgan Kaufmann.

Caserta, Pierre et O Zendra. Juillet 2011. « Visualization of the Static Aspects of Software : A Survey ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, n° 7, p. 913–933.

Chansler, Robert, Russell Bryant, Roy Bryant, Rosangela Canino-Koening, Francesco Cesarini, Eric Allman, Keith Bostic, et Titus Brown, 2011. *The Architecture of Open Source Applications*. New York, NY : Creative Commons Attribution.

Chen, C. 2005. « Top 10 unsolved information visualization problems ». *Computer Graphics and Applications, IEEE*, vol. 25, n° 4, p. 12-16.

Chevalier, Fanny, David Auber, et Alexandru Telea. 2007. « Structural analysis and visualization of C++ code evolution using syntax trees ». In *Ninth international workshop on Principles of software evolution : in conjunction with the 6th ESEC/FSE joint meeting*. (New York, NY, USA 2007), p. 90-97. ACM.

Chevalier, Fanny, Pierre Dragicevic, Anastasia Bezerianos, et Jean-Daniel Fekete. 2010. « Using Text Animated Transitions to Support Navigation in Document Histories ». In *Proc. ACM CHI*. p. 683–692.

Chidamber, S.R. et C.F. Kemerer. jun 1994. « A metrics suite for object oriented design ». *IEEE Transactions on Software Engineering*, vol. 20, n° 6, p. 476-493.

Cockburn, Alistair. Octobre 1996. « The interaction of social issues and software architecture ». *Commun*, vol. 39, n° 10, p. 40–46.

Collins, Christopher et Sheelagh Carpendale. 2007. « VisLink : Revealing relationships amongst visualizations ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, n° 6, p. 1192-1199.

Cottam, Joseph A., Andrew Lumsdaine, et Chris Weaver. 2012. « Watch this : A taxonomy for dynamic data visualization ». In *Proc. IEEE VAST*. p. 193–202.

D'Ambros, M., M. Lanza, et M. Lungu. 2009. « Visualizing Co-Change Information with the Evolution Radar ». *IEEE Transactions on Software Engineering*, vol. 35, n° 5, p. 720–735.

D'Ambros, Marco et Michele Lanza. 2009. « Visual software evolution reconstruction ». *J. Softw. Maint. Evol.*, vol. 21, n° 3, p. 217–232.

Denier, Simon et Houari Sahraoui. 10 2009. « Understanding the use of inheritance with visual patterns ». In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. p. 79-88. IEEE.

Di Battista, Giuseppe et Fabrizio Frati. Nov 2009. « Efficient C-Planarity Testing for Embedded Flat Clustered Graphs with Small Faces ». *Journal of Graph Algorithms and Applications*, vol. 13, n° 3, p. 349–378.

Di Battista, Giuseppe, Peter Eades, Roberto Tamassia, et Ioannis G. Tollis, 1999. *Graph Drawing : Algorithms for the Visualization of Graphs*. Upper Saddle River, N.J. : Prentice-Hall.

Diehl, Stephan, 2007. *Visualizing The Structure, Behaviour, And Evolution Of Software*. Germany : Springer-Verlag Berlin And Heidelberg Gmbh & Co. Kg, 199 p.

Diel, Stepahn, Carsten Görg, et Andreas Kerren. 2001. « Preserving the mental map using foresighted layout ». In *Proceedings of the 3rd Joint Eurographics-IEEE TCVG conference on Visualization*. p. 175-184. Eurographics Association.

Dinkla, Kasper, Michel A. Westenberg, et Jarke J. van Wijk. 2012. « Compressed Adjacency Matrices : Untangling Gene Regulatory Networks ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, n° 12, p. 2457-2466.

Ducasse, Stéphane et Michele Lanza. 1 2005. « The Class Blueprint : Visually Supporting the Understanding of Classes ». *IEEE Trans. Softw. Eng.*, vol. 31, n° 1, p. 75-90.

Ducasse, Stéphane, Michele Lanza, et Laura Ponisio. 2005. « Butterflies : A visual approach to characterize packages ». In *Software Metrics, 2005. 11th IEEE International Symposium*. p. 1-10. IEEE.

Ducasse, Stéphane, Damien Pollet, Mathieu Suen, Hani Abdeen, et Ilham Alloui. 2007. « Package surface blueprints : Visually supporting the understanding of package relationships ». In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*. p. 94-103. IEEE.

Eccles, Ryan, Thomas Kapler, Robert Harper, et William Wright. 10 2007. « Stories in GeoTime ». In *2007 IEEE Symposium on Visual Analytics Science and Technology*. p. 19-26. IEEE.

Eick, Stephen G., Joseph L. Steffen, et Jr. Eric E. Sumner. 1992. « Seesoft-A Tool for Visualizing Line Oriented Software Statistics ». *IEEE Trans. Softw. Eng.*, vol. 18, n° 11, p. 957-968.

Elmqvist, N. et J.-D. Fekete. 2010. « Hierarchical Aggregation for Information Visualization : Overview, Techniques, and Design Guidelines ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, n° 3, p. 439–454.

Elmqvist, N., Thanh-Nghi Do, H. Goodell, N. Henry, et J. Fekete. 2008. « ZAME : Interactive large-scale graph visualization ». In *Visualization Symposium, 2008. PacificVIS'08. IEEE Pacific*. p. 215-222. IEEE.

Erten, Cesim, Philip Harding, Stephen Kobourov, Kevin Wampler, et Gary Yee, 2004. *GraphAEL : Graph Animations with Evolving Layouts*, volume 2912 of *Lecture Notes in Computer Science*, p. 98-110. Springer Berlin / Heidelberg.

Farrugia, Michael et Aaron Quigley. Janvier 2011. « Effective Temporal Graph Layout : A Comparative Study of Animation versus Static Display Methods ». *Information Visualization*, vol. 10, n° 1, p. 47 –64.

Fayad, Mohamed et Douglas C. Schmidt. Octobre 1997. « Object-oriented application frameworks ». *Commun. ACM*, vol. 40, n° 10, p. 32–38.

Federico, Paolo, Wolfgang Aigner, Silvia Miksch, Florian Windhager, et Lukas Zenk. 2011. « A visual analytics approach to dynamic social networks ». In *Proc. I-KNOW*. p. 47 :1–47 :8. ACM.

Fekete, Jean-Daniel, David Wang, Niem Dang, Aleks Aris, et Catherine Plaisant. 2003. « Overlaying Graph Links on Treemaps ». In *Proceedings of IEEE Symposium on Information Visualization (InfoVis) Poster Compendium*. (Seattle, Washington 2003), p. 82–83.

Fischer, Michael et Harald Gall. 2006. « Evograph : A lightweight approach to evolutionary and structural analysis of large software systems ». In *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on*. p. 179-188. IEEE.

Frishman, Yaniv et Ayellet Tal. 2008. « Online dynamic graph drawing ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, n° 4, p. 727-740.

Fruchterman, Thomas MJ et Edward M. Reingold. 1991. « Graph drawing by force-directed placement ». *Software : Practice and experience*, vol. 21, n° 11, p. 1129-1164.

Gaertler, Marco et Dorothea Wagner. 2005. « A Hybrid Model for Drawing Dynamic and Evolving Graphs ». In *Proc. Symposium on Graph Drawing (GD)*. p. 189–200.

Gallagher, K., A. Hatch, et M. Munro. 2008. « Software Architecture Visualization : An Evaluation Framework and Its Application ». *IEEE Trans. on Software Engineering*, vol. 34, n° 2, p. 260–270.

Gamma, Erich et Kent Beck, 2003. *Contributing to Eclipse : Principles, Patterns, and Plugins*. Redwood City, CA, USA : Addison Wesley Longman Publishing Co., Inc.

Gamma, Erich, Richard Helm, Ralph Johnson, et John Vlissides, Novembre 1994. *Design Patterns : Elements of Reusable Object-Oriented Software*. éd. 1. Addison-Wesley Professional.

Gansner, Emden R., Yehuda Koren, et Stephen C. North. 2005. « Topological Fisheye Views for Visualizing Large Graphs ». *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 11, n° 4, p. 457–468.

Garlan, David et Mary Shaw. 1993. « An Introduction to Software Architecture ». *Advances in Software Engineering and Knowledge Engineering*, vol. 2, p. 1–39.

Gene Ontology Consortium. May 2000. « Gene ontology : tool for the unification of biology ». *Nature Genetics*, vol. 25, n° 1, p. 25–29.

German, Daniel M., Ahmed E. Hassan, et Gregorio Robles. 2009. « Change impact graphs : Determining the impact of prior codechanges ». *Information and Software Technology*, vol. 51, n° 10, p. 1394–1408.

Ghanam, Y. et S. Carpendale. 2008. « A survey paper on software architecture visualization ». *University of Calgary, Tech. Rep*.

Ghoniem, Mohammad, Jean-Daniel Fekete, et Philippe Castagliola. 2005. « On the readability of graphs using node-link and matrix-based representations : a controlled experiment and statistical analysis ». *Information Visualization*, vol. 4, n° 2, p. 114-135.

Girba, Tudor, Michele Lanza, et Stéphane Ducasse. 2005. « Characterizing the evolution of class hierarchies ». In *Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on*. p. 2-11. IEEE.

Graham, Hamish, Hong Yul Yang, et Rebecca Berrigan. 2004. « A solar system metaphor for 3D visualisation of object oriented software metrics ». In *Proceedings of the 2004 Australasian symposium on Information Visualisation-Volume 35*. p. 53-59. Australian Computer Society, Inc.

Grand, Mark, 2002. *Patterns in Java : A Catalog of Reusable Design Patterns Illustrated with Uml,Volume 1*. éd. 2nd. New York, NY, USA : John Wiley & Sons, Inc.

Greilich, Martin, Michael Burch, et Stephan Diehl. June 2009. « Visualizing the Evolution of Compound Digraphs with TimeArcTrees ». *Computer Graphics Forum*, vol. 28, n° 3, p. 975–982.

Griffin, Amy L, Alan M MacEachren, Frank Hardisty, Erik Steiner, et Bonan Li. Décembre 2006. « A Comparison of Animated Maps with Static Small-Multiple Maps for Visually Identifying Space-Time Clusters ». *Annals of the Association of American Geographers*, vol. 96, n° 4, p. 740–753.

Hadlak, Steffen, Hans-Jörg Schulz, et Heidrun Schumann. Décembre 2011. « In Situ Exploration of Large Dynamic Networks ». *IEEE TVCG*, vol. 17, n° 12, p. 2334–2343.

Hall, Richard, Karl Pauls, Stuart McCulloch, et David Savage, Avril 2011. *OSGi in Action : Creating Modular Applications in Java*. Manning Publications.

Harel, David. May 1988. « On Visual Formalisms ». *Communications of the ACM (CACM)*, vol. 31, n° 5, p. 514–530.

Hartigan, John A. 1975. « Printer graphics for clustering ». *Journal of Statistical Computation and Simulation*, vol. 4, n° 3, p. 187-213.

Heer, Jeffrey et George G. Robertson. 2007. « Animated Transitions in Statistical Data Graphics ». *IEEE TVCG*, vol. 13, n° 6, p. 1240–1247.

Heer, Jeffrey, Stuart K. Card, et James A. Landay. 2005. « Prefuse : A Toolkit for Interactive Information Visualization ». In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*. (Portland, Oregon 2005), p. 421–430.

Henry, N. et J.-D. Fekete. 2006. « MatrixExplorer : a Dual-Representation System to Explore Social Networks ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, n° 5, p. 677-684.

Henry, Nathalie. 2008. « Exploring Social Networks with Matrix-based Representations ». PhD thesis, Université Paris Sud, France, and University of Sydney, Australia.

Henry, Nathalie et Jean-Daniel Fekete. 2007. « MatLink : Enhanced Matrix Visualization for Analyzing Social Networks ». In *Proceedings of IFIP TC13 International Conference on Human-Computer Interaction (INTERACT)*. (Rio de Janeiro, Brazil 2007), p. 288–302.

Henry, Nathalie, Jean-Daniel Fekete, et Michael J. McGuffin. 2007. « NodeTrix : A Hybrid Visualization of Social Networks ». *IEEE TVCG*, vol. 13, n° 6, p. 1302–1309.

Henry, Nathalie, Anastasia Bezerianos, et Jean-Daniel Fekete. 2008. « Improving the Readability of Clustered Social Networks using Node Duplication ». *IEEE TVCG*, vol. 14, n° 6, p. 1317–1324.

Herman, Ivan, Guy Melançon, et M. Scott Marshall. January 2000. « Graph Visualization and Navigation in Information Visualization : A Survey ». *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 6, n° 1, p. 24–43.

Hinckley, Ken, Patrick Baudisch, Gonzalo Ramos, et Francois Guimbretiere. 2005. « Design and analysis of delimiters for selection-action pen gesture phrases in scriboli ». In *Proc. ACM CHI*. p. 451–460.

Hindle, A., Jiang Zhen Ming, W. Koleilat, M. W. Godfrey, et R. C. Holt. 2007. « YARN : Animating Software Evolution ». In *4th IEEE International Workshop on Visualizing Software for Understanding and Analysis VISSOFT*. p. 129–136.

Holten, Danny. 2006. « Hierarchical Edge Bundles : Visualization of Adjacency Relations in Hierarchical Data ». *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 12, n° 5, p. 741–748.

Holten, Danny et Jarke J. Van Wijk. 2008. « Visual Comparison of Hierarchically Organized Data ». *Computer Graphics Forum*, vol. 27, n° 3, p. 759–766.

Hurtado Alegría, Julio A., María Cecilia Bastarrica, et Alexandre Bergel. 2013. « Avispa : a tool for analyzing software process models ». *Journal of Software : Evolution and Process*.

Inselberg, Alfred, 2009. *Parallel coordinates : visual multidimensional geometry and its applications*. Springer.

Jacobson, Ivar, Grady Booch, et James Rumbaugh, 1999. *The unified software development process*. Reading, Mass : Addison-Wesley.

Javed, Waqas et Niklas Elmqvist. 2012. « Exploring the design space of composite visualization ». In *Pacific Visualization Symposium (PacificVis)*. p. 1-8. IEEE.

Johnson, Brian et Ben Shneiderman. 1991. « Tree-maps : A space-filling approach to the visualization of hierarchical information structures ». In *Proceedings of IEEE Visualization (VIS)*. (San Diego, CA 1991), p. 284–291.

Jürgensmann, Susanne et Hans-Jörg Schulz. 2010. « A Visual Survey of Tree Visualization ». In *Proceedings of IEEE Symposium on Information Visualization (InfoVis) Poster Compendium*. (Salt Lake City, Utah 2010).

Kaufmann, Michael et Dorothea Wagner, 2001. *Drawing Graphs : Methods and Models*. Berlin, New York : Springer.

Khan, Taimur, Henning Barthel, Achim Ebert, et Peter Liggesmeyer. 2012. « Visualization and Evolution of Software Architectures ». In *Visualization of Large and Unstructured Data Sets : Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011*. (Dagstuhl, Germany 2012), p. 25–42. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Knight, Claire et Malcolm Munro. 1999. « Comprehension with [in] virtual environment visualisations ». In *Proceedings of the Seventh International Workshop on Program Comprehension*. p. 4-11. IEEE.

Komlodi, A., A. Sears, et E. Stanziola. 2004. ». In *Information visualization evaluation review*. ISRC Tech. Report, Dept. of Information Systems, UMBC.

Kosara, R., C. G. Healey, V. Interrante, D. H. Laidlaw, et C. Ware. 2003. « Visualization viewpoints ». *Computer Graphics and Applications, IEEE*, vol. 23, n° 4, p. 20-25.

Kumar, Gautam et Michael Garland. 2006. « Visual exploration of complex time-varying graphs ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, n° 5, p. 805-812.

Kurtenbach, Gordon et William Buxton. 1993. « The Limits of Expert Performance Using Hierarchic Marking Menus ». In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*. (Amsterdam, The Netherlands 1993), p. 482–487.

Langelier, G., H. Sahraoui, et P. Poulin. 2008. « Exploring the evolution of software quality with animated visualization ». In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2008)*. p. 13–20.

Langelier, Guillaume, Houari Sahraoui, et Pierre Poulin. 2005. « Visualization-based analysis of quality for large-scale software systems ». In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. p. 214-223. ACM.

Lanza, Michele. 2001. « The evolution matrix : Recovering software evolution using software visualization techniques ». In *Proceedings of the 4th international workshop on principles of software evolution*. p. 37-42. ACM.

Lanza, Michele et Stéphane Ducasse. 2001. « A categorization of classes based on the visualization of their internal structure : the class blueprint ». *SIGPLAN Not.*, vol. 36, n° 11, p. 300-311.

Larman, C. 2001. « Protected variation : the importance of being closed ». *IEEE Software*, vol. 18, n° 3, p. 89–91.

Larman, Craig, 2005. *Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development*. éd. 3rd. Upper Saddle River, N.J. : Prentice Hall PTR.

Lee, Bongshin, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, et Nathalie Henry. 2006. « Task Taxonomy for Graph Visualization ». In *Proceedings of AVI workshop BEyond time and errors : novel evaLuation methods for Information Visualization (BELIV)*. (Venice, Italy 2006), p. 1–5. ACM.

Lieberherr, Karl J., Ian Holland, et Arthur J. Riel. September 1988. « Object-oriented programming : An objective sense of style ». In *oopsla*. (San Diego, CA 1988), p. 323-334.

MacKenzie, I. Scott. 1992. « Fitts' law as a research and design tool in human-computer interaction ». *Human-Computer Interaction*, vol. 7, n° 1, p. 91–139.

Mackinlay, J. D., P. Hanrahan, et C. Stolte. 2007. « Show Me : Automatic Presentation for Visual Analysis ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, n° 6, p. 1137-1144.

Mäkinen, Erkki et Harri Siirtola. 2000. « Reordering the Reorderable Matrix as an Algorithmic Problem ». In *International Conference on the Theory and Application of Diagrams*. (Edinburgh, UK 2000), p. 453–468.

Marghescu, Dorina, Mikko Rajanen, et Barbro Back. 2004. « Evaluating the quality of use of visual data-mining tools ». In *Proc. 11th European Conference on Information Technology Evaluation*. p. 239-250.

Mazza, Riccardo, 2009. *Introduction to Information Visualization*. Springer Publishing Company, Incorporated, 139 p.

McConnell, Steve, 2004. *Code Complete, Second Edition*. Redmond, WA, USA : Microsoft Press.

McGuffin, Michael J. 2012. « Simple algorithms for network visualization : A tutorial ». *Tsinghua Science and Technology*, vol. 17, n° 4, p. 383-398.

McGuffin, Michael J. et Jean-Marc Robert. 2010. « Quantifying the Space-Efficiency of 2D Graphical Representations of Trees ». *Information Visualization*, vol. 9, n° 2, p. 115–140.

McNair, A., D. M. German, et J. Weber-Jahnke. 2007. « Visualizing Software Architecture Evolution Using Change-Sets ». In *14th Working Conference on Reverse Engineering*. p. 130–139.

Mens, Tom, Jim Buckley, Matthias Zenger, et Awais Rashid. 2003. « Towards a taxonomy of software evolution ». In *Proc. Workshop on Unanticipated Software Evolution*. p. 1-18.

Mens, Tom, Michel Wermelinger, Stéphane Ducasse, Serge Demeyer, Robert Hirschfeld, et Mehdi Jazayeri. 2005. « Challenges in software evolution ». In *Eighth International Workshop on Principles of Software Evolution*. p. 13-22. IEEE.

Meyer, Bertrand, 1988. *Object-oriented software construction*. NY : Prentice Hall.

Microsoft, 2012. *C# Reference*. Microsoft.

Mueller, Christopher, Benjamin Martin, et Andrew Lumsdaine. 2007. « A comparison of vertex ordering algorithms for large graph visualization ». In *6th International Asia-Pacific Symposium on*. p. 141-148. IEEE.

Munzner, Tamara. 2009. « A Nested Model for Visualization Design and Validation ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, n° 6, p. 921-928.

Neumann, Petra, Stefan Schlechtweg, et Sheelagh Carpendale. 2005. « Arctrees : Visualizing relations in hierarchical data ». In *Proceedings of the Seventh Joint Eurographics/IEEE VGTC conference on Visualization*. p. 53-60. Eurographics Association.

North, C. Juin 2006. « Toward measuring visualization insight ». *IEEE Computer Graphics and Applications*, vol. 26, n° 3, p. 6 –9.

North, Stephen C. 1996. « Incremental layout in DynaDAG ». In *Graph Drawing*. p. 409-418. Springer.

Novais, Renato Lima, André Torres, Thiago Souto Mendes, Manoel Mendonça, et Nico Zazworka. 2013. « Software Evolution Visualization : A Systematic Mapping Study ». *Information and Software Technology. (Article en impression)*.

Ogawa, M. et KL Ma. 2009. « code_swarm : A Design Study in Organic Software Visualization ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, n° 6, p. 1097-1104.

Parnas, David Lorge. 1971. « Information Distribution Aspects of Design Methodology ». In *IFIP Congress (1)*. p. 339–344.

Parnas, David Lorge. 1994. « Software aging ». In *Proceedings of the 16th international conference on Software engineering*. (Sorrento, Italy 1994), p. 279–287. IEEE Computer Society Press.

Perry, Dewayne E. et Alexander L. Wolf. Octobre 1992. « Foundations for the study of software architecture ». *SIGSOFT Softw. Eng. Notes*, vol. 17, n° 4, p. 40–52.

Pinzger, Martin, Harald Gall, Michael Fischer, et Michele Lanza. 2005. « Visualizing multiple evolution metrics ». In *Proceedings of the ACM symposium on Software visualization*. (New York, NY, USA 2005), p. 67-75. ACM.

Plaisant, Catherine. 2004. « The challenge of information visualization evaluation ». In *Proceedings of the working conference on Advanced visual interfaces*. (New York, NY, USA 2004), p. 109–116. ACM.

Plaisant, Catherine, Jesse Grosjean, et Benjamin B. Bederson. 2002. « SpaceTree : Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation ». In *Proc. IEEE InfoVis*. p. 57–64.

Pretorius, A. Johannes et Jarke J. van Wijk. 2006. « Visual Analysis of Multivariate State Transition Graphs ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, n° 5, p. 685–692.

Purchase, Helen et Amanjit Samra. 2008. Extremes are better : Investigating mental map preservation in dynamic graphs. *Diagrammatic Representation and Inference*, volume 5223 of *LNCS*, p. 60–73. Springer.

Purchase, Helen C., Eve Hoggan, et Carsten Görg. 2006. « How Important Is the "Mental Map" ?-An Empirical Investigation of a Dynamic Graph Layout Algorithm ». In *Proc. Symposium on Graph Drawing (GD)*. p. 184–195.

Ramil, Juan F. et Meir M. Lehman. 2000. « Metrics of software evolution as effort predictors-a case study ». In *Software Maintenance, 2000. Proceedings. International Conference on*. p. 163-172. IEEE.

Ratzinger, Jacek, Michael Fischer, et Harald Gall. 2005. « Improving evolvability through refactoring ». In *ACM SIGSOFT Software Engineering Notes*. p. 1-5. ACM.

Reda, Khairi, Chayant Tantipathananandh, Andrew Johnson, Jason Leigh, et Tanya Berger-Wolf. Juin 2011. « Visualizing the Evolution of Community Structures in Dynamic Social Networks ». *Computer Graphics Forum*, vol. 30, n° 3, p. 1061–1070.

Reenskaug, Trygve. 5 1979. *Thing-model-view-editor–an example from a planning system*.

Reinhold, Mark. 19 Avril 2011. « Java Module-System Requirements – DRAFT 12 ».

Roberts, Jonathan C. 2007. « State of the art : Coordinated & multiple views in exploratory visualization ». In *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization*. p. 61-71. IEEE.

Rosvall, Martin et Carl T. Bergstrom. Janvier 2010. « Mapping Change in Large Networks ». *PLoS ONE*, vol. 5, n° 1.

Rufiange, Sébastien, Michael J. McGuffin, et Christopher Fuhrman. 2009. « Visualisation hybride des liens hiérarchiques incorporant des treemaps dans une matrice d'adjacence ». In *Conférence sur l'Interaction Homme-Machine (IHM)*. (Grenoble, France 2009), p. 51–54.

Rufiange, Sébastien, Michael J. McGuffin, et Christopher P. Fuhrman. 2012. « TreeMatrix : A Hybrid Visualization of Compound Graphs ». *Computer Graphics Forum*, vol. 31, n° 1, p. 89-101.

Sallaberry, Arnaud, Chris Muelder, et Kwan-Liu Ma. 2013. Clustering, visualizing, and navigating for large dynamic graphs. *Graph Drawing*, volume 7704 of *LNCS*, p. 487-498. Springer.

Sander, Georg. 96. *Layout of compound directed graphs*. Technical report. Universität des Saarlandes, Saarbrücken, Germany.

Sangal, Neeraj, Ev Jordan, Vineet Sinha, et Daniel Jackson. 2005. « Using dependency models to manage complex software architecture ». In *Proc. of the 20th annual ACM SIGPLAN conference on OOPSLA*. p. 167–176. ACM.

Saraiya, P., C. North, et K. Duca. Août 2005a. « An insight-based methodology for evaluating bioinformatics visualizations ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, n° 4, p. 443 –456.

Saraiya, Purvi, Peter Lee, et Chris North. 2005b. « Visualization of graphs with associated timeseries data ». In *Proc. IEEE InfoVis*. p. 225–232.

Schulz, Hans-Jorg . J. 2011. « Treevis.net : A Tree Visualization Reference ». *IEEE Computer Graphics and Applications*, vol. 31, n° 6, p. 11-15.

Shen, Zeqian et Kwan-Liu Ma. 2007. « Path Visualization for Adjacency Matrices ». In *Proceedings of Eurographics/IEEE-VGTC Symposium on Visualization (EuroVis)*. (Norrköping, Sweden 2007), p. 83–90.

Shneiderman, Ben. 1996. « The Eyes Have It : A Task by Data Type Taxonomy for Information Visualizations ». In *IEEE Symposium on Visual Languages*. p. 336-343. IEEE Computer Society.

Shneiderman, Ben et Catherine Plaisant. 2006. ». In *Strategies for evaluating information visualization tools : multi-dimensional in-depth long-term case studies*. (Venice, Italy 2006), p. 1-7. ACM.

Sindre, Guttorm, Bjørn Gulla, et Håkon G. Jokstad. 1993. « Onion Graphs : Aesthetics and Layout ». In *Proceedings of IEEE Symposium on Visual Languages (VL)*. (Bergen , Norway 1993), p. 287–291.

Stasko, John. 2000. « An evaluation of space-filling information visualizations for depicting hierarchical structures ». *Int. J. Hum.-Comput. Stud.*, vol. 53, n° 5, p. 663-694.

Stasko, John et Eugene Zhang. 2000. « Focus+ context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations ». In *IEEE Symposium on Information Visualization*. p. 57-65. IEEE.

Sugiyama, Kozo et Kazuo Misue. July/August 1991. « Visualization of Structural Information : Automatic Drawing of Compound Digraphs ». *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21, n° 4, p. 876–892.

Sugiyama, Kozo, Shojiro Tagawa, et Mitsuhiko Toda. February 1981a. « Methods for Visual Understanding of Hierarchical System Structures ». *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, n° 2, p. 109–125.

Sugiyama, Kozo, Shojiro Tagawa, et Mitsuhiko Toda. 1981b. « Methods for visual understanding of hierarchical system structures ». *IEEE Transactions on Systems, Man and Cybernetics*, vol. 11, n° 2, p. 109-125.

Sutcliffe, A. G., M. Ennis, et J. Hu. 2000. « Evaluating the effectiveness of visual user interfaces for information retrieval ». *Int. J. Hum.-Comput. Stud.*, vol. 53, n° 5, p. 741-763.

Taibi, Toufik, Février 2007. *Design Pattern Formalization Techniques*. Idea Group Inc (IGI).

Taylor, Christopher MB et Malcolm Munro. 2002. « Revision towers ». In *Proceedings of the First International Workshop on Visualizing Software for Understanding and Analysis*. p. 43-50. IEEE.

Telea, Alexandru et David Auber. Mai 2008. « Code Flows : Visualizing Structural Evolution of Source Code ». *Computer Graphics Forum*, vol. 27, n° 3, p. 831–838.

Thomas, J. J. et K. A. Cook. 1 2006. « A visual analytics agenda ». *Computer Graphics and Applications, IEEE*, vol. 26, n° 1, p. 10-13.

Tversky, Barbara, Julie Bauer Morrison, et Mireille Betrancourt. 2002. « Animation : can it facilitate ? ». *International Journal of Human-Computer Studies*, vol. 57, n° 4, p. 247–262.

Vanbrabant, Robbie, Avril 2008. *Google Guice : Agile Lightweight Dependency Injection Framework*. éd. 1. Apress.

Vanya, Adam, Rahul Premraj, et Hans van Vliet. Avril 2012. « Resolving unwanted couplings through interactive exploration of co-evolving software entities – An experience report ». *Information and Software Technology*, vol. 54, n° 4, p. 347–359.

Viau, C., M. J. McGuffin, Y. Chiricota, et I. Jurisica. 2010. « The FlowVizMenu and Parallel Scatterplot Matrix : Hybrid Multidimensional Visualizations for Network Exploration ». *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, n° 6, p. 1100-1108.

Voinea, Lucian et Alexandru Telea. 2007. « Visual data mining and analysis of software repositories ». *Computers & Graphics*, vol. 31, n° 3, p. 410–428.

Voinea, Lucian, Alex Telea, et Jarke J. van Wijk. 2005. « CVSscan : visualization of code evolution ». In *Proceedings of the 2005 ACM symposium on Software visualization*. (St. Louis, Missouri 2005), p. 47–56. ACM.

Voinea, S. L. et Alexandru Telea. 2006. « Cvsgrab : Mining the history of large software projects ». In *Proceedings of the Eighth Joint Eurographics/IEEE VGTC conference on Visualization*. p. 187-194. Eurographics Association.

von Landesberger, T., A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, et D. W. Fellner. 2010. « Visual analysis of large graphs ». In *EuroGraphics : State of the Art Report*. (Norrköping, Sweden 2010).

von Landesberger, T., A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J. D Fekete, et D. W. Fellner. 9 2011. « Visual Analysis of Large Graphs : State of the Art and Future Research Challenges ». *Computer Graphics Forum*, vol. 30, n° 6, p. 1719-1749.

Ware, Colin, 2000. *Information visualization : perception for design*. Morgan Kaufmann Publishers Inc., 438 p.

Wattenberg, Martin. 2002. « Arc Diagrams : Visualizing Structure in Strings ». In *Proc. IEEE InfoVis*. p. 110–116.

Wattenberg, Martin. 2005. « A Note on Space-Filling Visualizations and Space-Filling Curves ». In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*. (Minneapolis, Minnesota 2005), p. 181–186.

Weissgerber, Peter, Mathias Pohl, et Michael Burch. 2007. « Visual Data Mining in Software Archives to Detect How Developers Work Together ». In *Proceedings of the Fourth International Workshop on Mining Software Repositories*. (Washington, DC, USA 2007), p. 9–. IEEE Computer Society.

Wettel, R. et M. Lanza. 2007. « Program Comprehension through Software Habitability ». In *Proceedings of the 15th IEEE International Conference on Program Comprehension*. p. 231–240.

Wettel, Richard et Michele Lanza. 2008a. « Visual Exploration of Large-Scale System Evolution ». In *Proceedings of the 15th Working Conference on Reverse Engineering*. p. 219-228. IEEE Computer Society.

Wettel, Richard et Michele Lanza. 2008b. « Visually localizing design problems with disharmony maps ». In *Proceedings of the 4th ACM symposium on Software visualization*. (Ammersee, Germany 2008), p. 155–164. ACM.

Windhager, Florian, Lukas Zenk, et Paolo Federico. 2011. « Visual Enterprise Network Analytics - Visualizing Organizational Change ». *Procedia - Social and Behavioral Sciences*, vol. 22, p. 59 - 68.

Wirfs-Brock, R., B.Wilkerson, et L. Wiener, 1990. *Designing Object-Oriented Software*. Prentice-Hall.

Xinrong, Xie, Poshyvanyk Denys, et Marcus Andrian. 2006. « Visualization of CVS Repository Information ». In *Proceedings of the 13th Working Conference on Reverse Engineering*. p. 231–242.

Yi, Ji Soo, Niklas Elmqvist, et Seungyoon Lee. 2010. « TimeMatrix : Analyzing Temporal Social Networks Using Interactive Matrix-Based Visualizations ». *International Journal of Human-Computer Interaction*, vol. 26, n° 11-12, p. 1031-1051.

Zaman, Loutfouz, Ashish Kalra, et Wolfgang Stuerzlinger. 2011. « The Effect of Animation, Dual View, Difference Layers, and Relative Re-Layout in Hierarchical Diagram Differencing ». In *Proc. Graphics Interface (GI)*. p. 183–190.

Zhao, Shengdong, Michael J. McGuffin, et Mark H. Chignell. 2005. « Elastic Hierarchies : Combining Treemaps and Node-Link Diagrams ». In *Proc. IEEE InfoVis*. p. 57–64.