SEAMAIL

Newsletter from Software Engineers Association

Vol. 14, Number 7 July, 2005

目　次

ソフトウェア技術者協会
Software Engineers Asociation

ソフトウェア技術者協会 (SEA) は，ソフトウェアハウス，コンピュータメーカ，計算センタ，エンドユーザ，大学，研究所など，それぞれ異なった環境に置かれているソフトウェア技術者または研究者が，そうした社会組織の壁を越えて，各自の経験や技術を自由に交流しあうための「場」として，1985年 12月に設立されました．

その主な活動は，機関誌 SEAMAIL の発行，支部および研究分科会の運営，セミナー／ワークショップ／シンポジウムなどのイベントの開催，および内外の関係諸団体との交流です．発足当初約 200人にすぎなかった会員数もその後増加し，現在，北は北海道から南は沖縄まで，400 余名を越えるメンバーを擁するにいたりました．法人賛助会員も 18社を数えます．支部は，東京以外に，関西，横浜，名古屋，九州，広島，東北の各地区で設立されており，その他の地域でも設立準備をしています．分科会は，東京，関西，名古屋で，それぞれいくつかが活動しており，その他の支部でも，月例会やフォーラムが定期的に開催されています．

「現在のソフトウェア界における最大の課題は，技術移転の促進である」といわれています．これまでわが国には，そのための適切な社会的メカニズムが欠けていたように思われます． SEA は，そうした欠落を補うべく，これからますます活発な活動を展開して行きたいと考えています．いままで日本にはなかったこの新しいプロフェッショナル・ソサイエティの発展のために，ぜひとも，あなたのお力を貸してください．

ＳＥＡＭＡＩＬ Ｖｏｌ．１４，Ｎｏ．７ ２００５年７月２０日発行 編集人 岸田 孝一
発行人 ソフトウェア技術者協会（ＳＥＡ）
〒１６０-０００４ 東京都新宿区四谷３－１２ 丸正ビル５Ｆ
T: 03-3356-1077 F: 03-3356-1072 E-mail: sea@sea.or.jp URL: http://www.sea.jp
印刷所 市田印刷株式会社 〒１１４-００１４ 東京都北区田端２－３－２５
定価 ５００円 （禁無断転載）

編集部から

☆

当初，この号はデザインワークショップ報告を予定していたのですが，途中に SS2005 in 富山がはさまってしまいました．

☆☆

SEA は時代の流れを先取りして，SS の Proceedings は 2 年前から紙ではなく CD に変わっています．その内容は添付した CD をお読みください．

☆☆☆

CD に含まれていない情報として，この号には，基調講演，招待講演，および SS Pre-Forum in 東京で使われた Presentation Slide のコピーと，SS 特別講演のメモを収録しました．

☆☆☆☆

次号は 2 月末に行われたデザインワークショップの報告です．

☆☆☆☆☆

基調講演
「ソフトウェア開発における協調作業支援の試み」
~分散型構成管理支援ツール Palantir を例として~

Andre van der Hoek (University of California, Irvine)

講師紹介

Andre van der Hoek (University of California, Irvine)
オランダ生れ．コロラド大学で PhD を取得した気鋭のソフトウェア工学研究者．
専門は構成管理，アーキテクチャ，プロダクトライン．
http://www.ics.uci.edu/~andre/

要旨

ソフトウェア工学ツールが基本的に抱えているパラドクスは，それらのツールがソフトウェア開発という協調作業を支援する使命を持っているにも関わらず，それぞれの個人やグループの仕事が他とは独立に行われているということである．

現存するツールのほとんどは，対象とする作業やその時間をはっきりと他から独立したプロセス・ステップに切り離すかたちになっている．そうしたアプローチは，人間の活動がそれぞれ個別に規定でき，一定の周期でそれらの同期を取ることが容易だという，誤った仮定にもとづいている．

この講演で，わたしは，連続的な協調支援を目指す１つの新しいアプローチを提案する．それは，ソフトウェア開発者たちが，同時並行的に進行している関連作業についての"Awareness Information（自覚情報)" を共有できるようにしたプロセス指向のアプローチである．このアプローチは，ソフトウェア開発プロセスの性格によくマッチしており，個々の開発者たちが，予想されるトラブルを避けるべくそれぞれの仕事を調整するのに十分な柔軟性を持った Awareness Information を提供することが可能である．

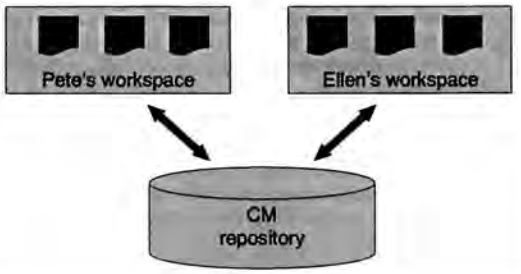
講演の初めに，わたしは，協調作業のためにツール群を組織する上での新しいフレームワークの歴史について簡単に述べる．次に，連続的協調の考え方について述べ，あわせてわれわれがその概念を Palantir と呼ばれるツールでどのように実現したかを説明する．このツールは，Awareness にもとづく作業空間における構成管理を支援している．そして最後に，われわれがいま研究開発中のいくつかのプロトタイプ・システムを紹介する．それらは並行作業の可視化を支援し，ソフトウェア設計における連続的協調を目的としたものである．

Continuous Coordination: Bridging Formal and Informal Coordination with Palantír

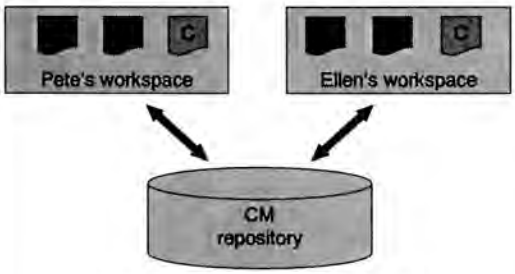
André van der Hoek
Donald Bren School of Information and Computer Sciences
Department of Informatics
andre@ics.uci.edu

June 2005

A Typical Development Scenario

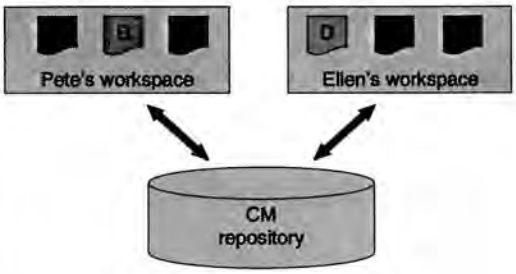


Direct Conflicts



Conflicting changes to the *same* artifact

Indirect Conflicts



Conflicting changes to *different* artifacts

Traditional CM Approaches

	Coordination mechanism	*Direct conflicts*	*Indirect conflicts*
Pessimistic	Locking before changes are made	Avoided, at the expense of project delays	Not addressed
Optimistic	Automated merging after changes have been made	Resolved, except for overlapping changes	Not addressed

Key Observations

- A CM workspace in reality provides two kinds of isolation:
  - good isolation
    - shields developers from parallel changes to artifacts
  - bad isolation
    - hides knowledge of what artifacts other developers are changing
- Opportunities for breaking isolation are limited
  - based on repository information only
  - initiated by developer only
  - addressing direct conflicts only
- In essence, a specific form of coordination is enforced and limited by the *process* underlying the CM system

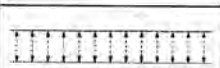
Awareness

- An alternative approach to coordination based on instant information sharing
- Relies on developers receiving the right information to self-coordinate
- In the extreme, it is "what you see is what I see"
  - shared editors
- In less extreme and more typical cases, it typically involves automated e-mails or annotations informing developers of other developer's actions
  - BSCW
  - Tukan

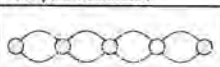
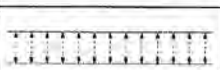
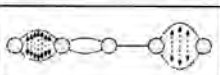
Awareness in Configuration Management

- Encodes an oft-used convention
  - send e-mail upon check out and upon check in
- Effective in keeping developers informed of when to integrate / synchronize with newly checked in code
- Completely inadequate in supporting coordination
  - does not avoid conflicts
  - does not support (direct or indirect) conflict detection
  - does not fundamentally alter the behavior of the developer in neither the pessimistic nor the optimistic approach
- Also exhibits the problem of information overload

Where We Stand Today

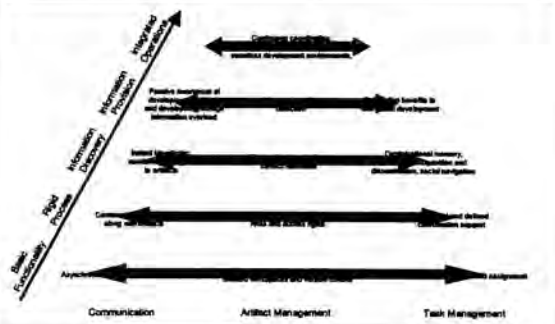
	Conceptual Visualization	Strengths	Weaknesses
Formal Coordination		**Scalable; Insulation from other activities; Control; Group-first**	**Reconciliation problems; Insulation becomes isolation**
Informal Coordination		**Flexible; Promotes synergy; Raises awareness; User-first**	**Not scalable; Requires human-intermediation**

Our Proposal: Continuous Coordination

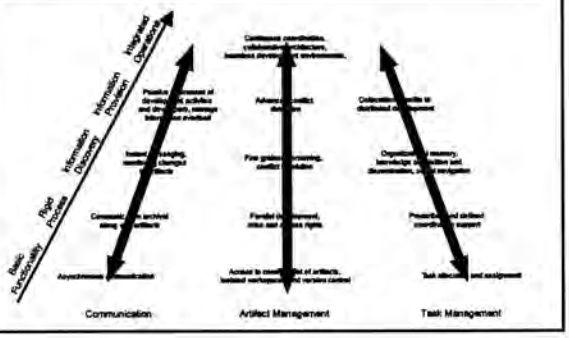
	Conceptual Visualization	Strengths	Weaknesses
Formal Coordination		**Scalable; Insulation from other activities; Control; Group-first**	**Reconciliation problems; Insulation becomes isolation**
Informal Coordination		**Flexible; Promotes synergy; Raises awareness; User-first**	**Not scalable; Requires human-intermediation**
Continuous Coordination		**?**	**?**

An Alternative View

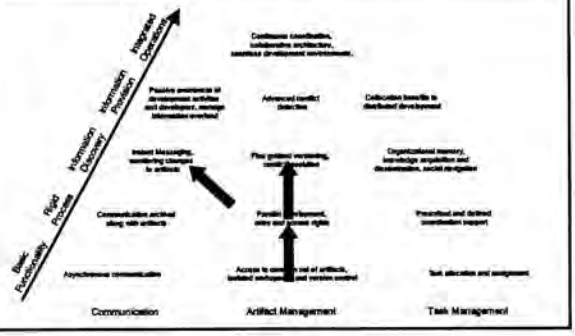
An Alternative View



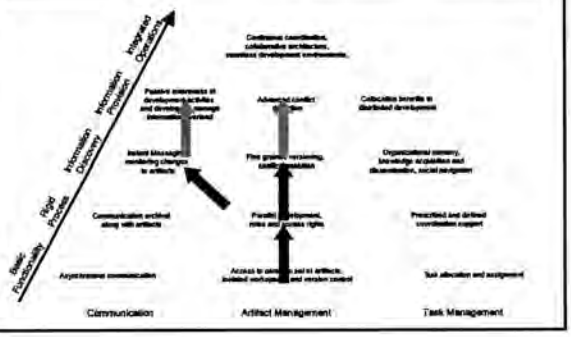
An Alternative View



Back to Configuration Management



Our Work



Objective

- To support continuous coordination by allowing "good isolation" (workspace insulation) but breaking "bad isolation" (workspace isolation)
  - in essence, combining the formal process-based check-out / check-in with relevant information sharing to create awareness of ongoing parallel activities
  - which artifacts are being changed by whom?
  - what is the severity of the changes?
- To, thereby, move earlier the point at which potential direct and indirect conflicts can be detected and resolved

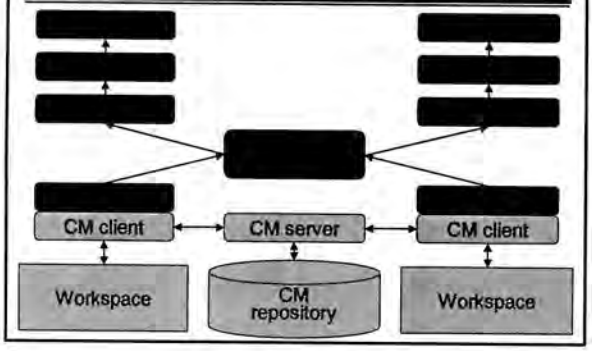
Approach

- Share information from workspace to workspace
  - collect
  - distribute
  - organize
  - present
- To do so automatically and continuously
  - push information

Design Constraints

- Non-obtrusiveness
- Scalability
- Flexibility
- Configurability

Palantír Architecture



Types of Events

Populated	Artifact has been placed in a workspace
Unpopulated	Artifact has been removed from a workspace
Synchronized	Artifact has been synchronized with repository
ChangesInProgress	Artifact has changed in a workspace
ChangesCommitted	Artifact has been stored in repository
ChangesReverted	Artifact has been returned to its original state
Added	New artifact has been added to workspace
Removed	Artifact has been removed from workspace
Renamed	Artifact has been renamed
Moved	Artifact has been moved from one artifact to another
SeverityChanged	Severity of changes to an artifact has changed

Typical Sequence of Events

Pessimistic
- Populated
- ChangesInProgress
- SeverityChanged
- SeverityChanged
- ...
- ChangesCommitted
- ChangesInProgress
- SeverityChanged
- SeverityChanged
- ...
- ChangesCommitted
- UnPopulated

Optimistic
- Populated
- ChangesInProgress
- SeverityChanged
- SeverityChanged
- ...
- ChangesCommitted
- ChangesInProgress
- SeverityChanged
- SeverityChanged
- ...
- ChangesCommitted
- UnPopulated

Event Wrapper

- Tasks
  - intercept workspace activity
  - interpret workspace activity
  - determine whether the activity is relevant to Palantír
  - if relevant, gather information regarding the activity
  - construct and emit one or more events
- Unique per CM system
  - wrap command line
  - use standard facilities (triggers, virtual file system)
- Can and should remain unobtrusive

Internal State

- Tasks
  - maintain overview of activities in both the local and remote workspaces
  - subscribe to relevant workspace activities
    - those pertaining to the artifacts in the local workspace but occurring in other workspaces
  - bootstrap new workspaces with information on existing workspaces
- Same for every CM system
- Always remains unobtrusive

Extractor

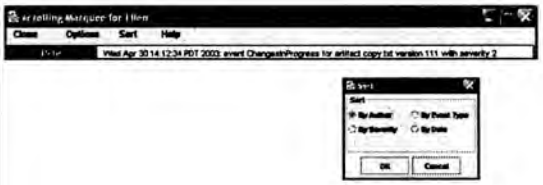
- Tasks
  - further narrow down the events to be viewed
    - set of workspaces
    - set of authors
    - set of artifacts
    - ...
- Same for every CM system
- One-time obtrusive

Extractor

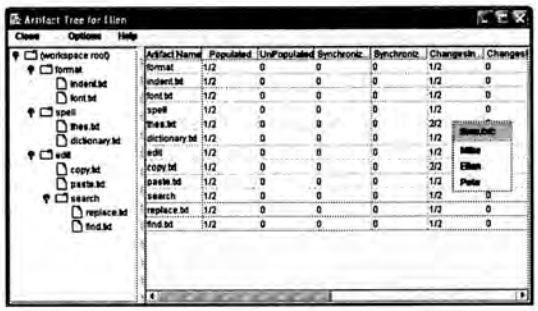
Visualization

- Tasks
  - organize and display events
- Same for every CM system
- Varying degrees of obtrusiveness
  - ticker tape
  - tabular
  - explorer
  - fully graphical
  - Eclipse

Ticker Tape

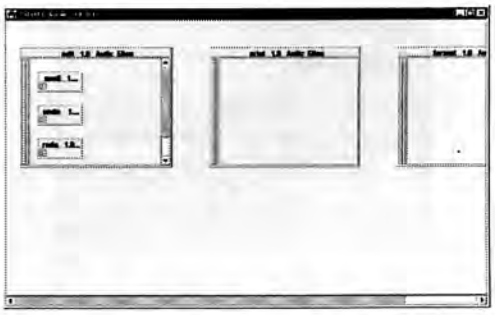


Tabular

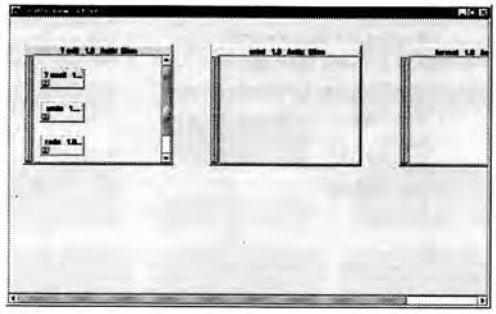


Explorer

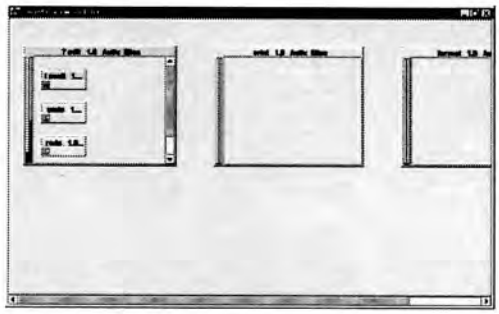
Populating a Workspace



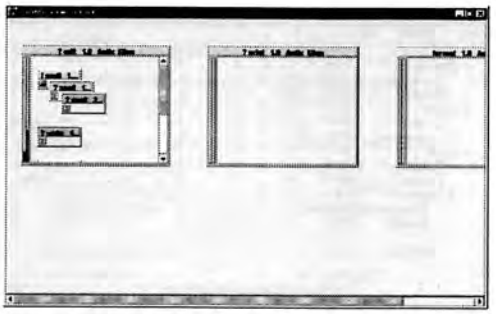
Making Changes in the Workspace



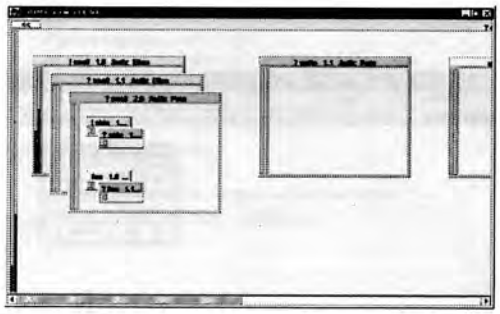
Committing Changes



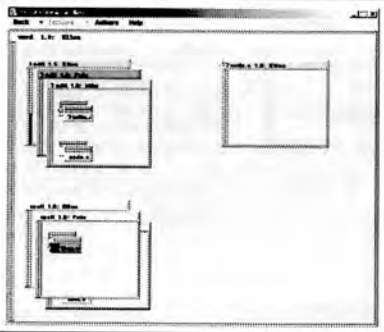
More Changes (by Other Developers)



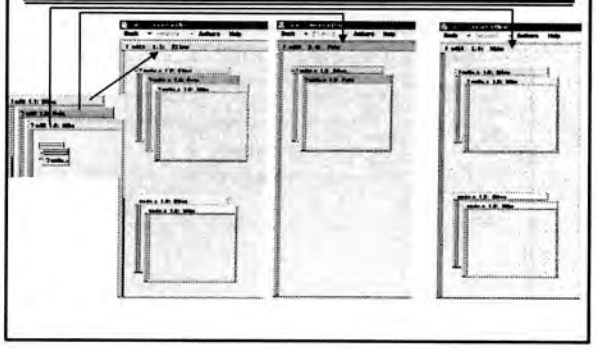
Etc., Etc., Etc...



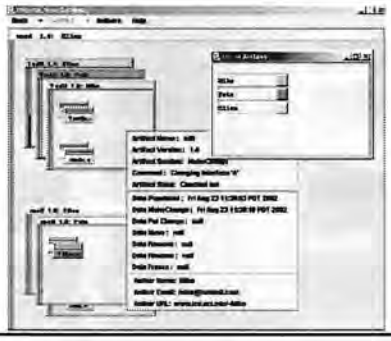
Removing and Moving Artifacts



Critical Feature: Pair-Wise Comparisons



Metadata



Eclipse Integation

Severity

- Currently calculated as the percentage of lines of code that have changed
  - simple
  - not completely accurate
    - 1 line change to an interface
    - 1000 lines renaming a variable
- Work in progress
  - other severity measures
  - impact measures

Scalability & Information Overload

- Application
  - manage only relevant artifacts
    - artifacts present in "my" workspace
    - leverages event service filtering
  - internal data structure versus visualization
- User cognition
  - pair-wise comparisons
  - stack shows linear evolution in time
  - filter data per user criteria
  - sorting of artifacts per severity / date

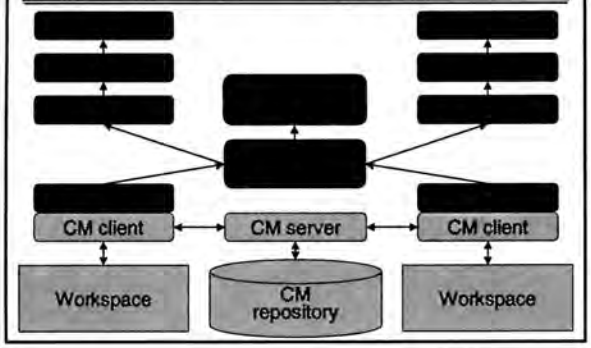
Integration Experience

- RCS
  - pessimistic CM system
  - 500 lines of Java code
  - single day
- CVS
  - optimistic CM system
  - 500 lines of Java code
  - single day
- JSVN
  - optimistic CM system
  - 500 lines of Java code
  - few days
- Eclipse
  - several days

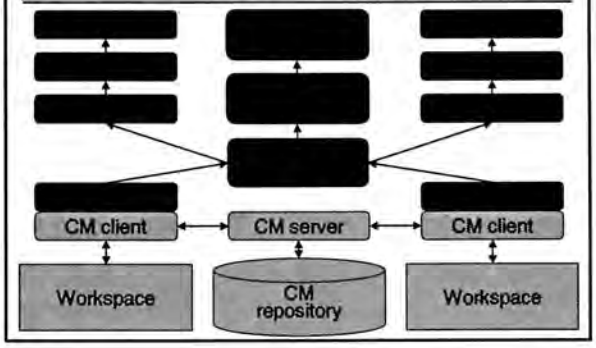
User Studies

- A first user study is currently being set up and planned with a specific CM system vendor
  - internal trials by the vendor
  - medium-sized group will use Palantír with their internal product
  - compare average number of conflicts before and after
  - qualitative feedback from individual developers

Opportunity: What If We Capture Events?



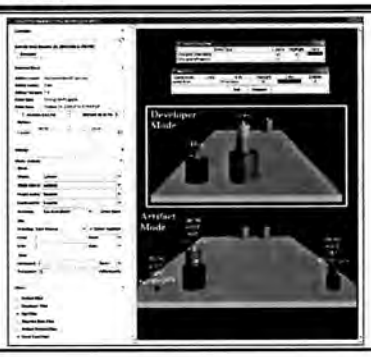
Enhanced Palantír Architecture



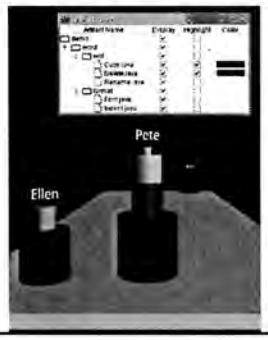
Specific Uses of Activity Viewer

- Visualize entire projects
- Replay a history of events
- Begin understanding projects and how they operate
- Recognize different types of developers
- Project scheduling

Activity Viewer



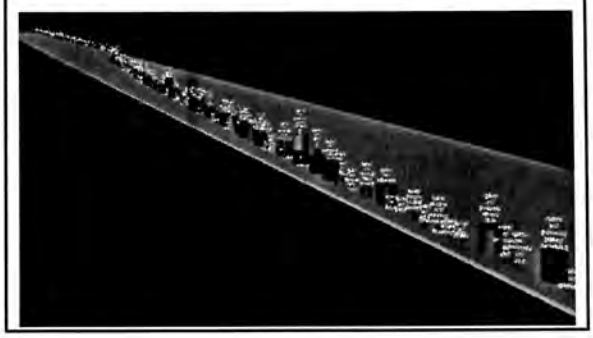
Highlighting



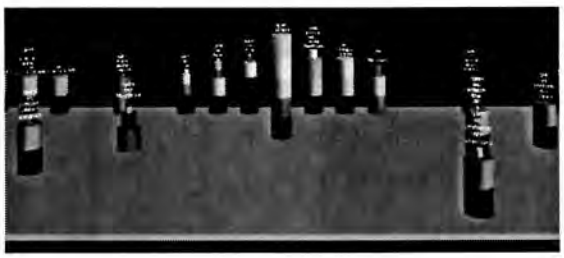
Distance is Age

Rotation Allows Different Viewpoints

Larger Projects: Time Ordered



Larger Projects: Most Activity Ordered



Larger Projects: Developer View

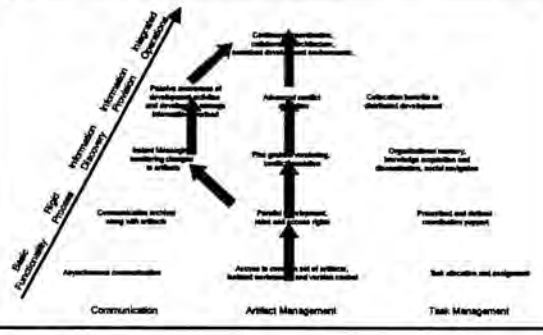
Same Project Ordered by Time

Committed versus Not Committed

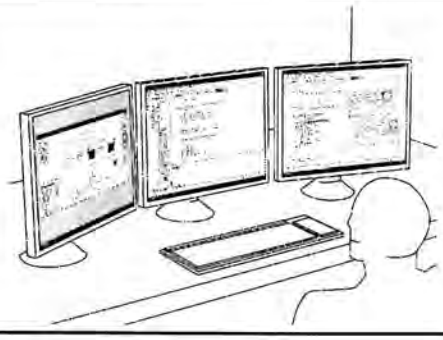
Applied to GAIM, jEdit, and Argo/UML

- Simulated the archives
  - demonstrated scalability
  - demonstrated usefulness
  - filters are a must
- Interesting patterns
  - core developers
  - core developer "overtaken" by others
  - lots of people on a project, but most are working on pictures, not code
  - highly active artifacts
  - ...
- Much more analysis to be done
- Planned: viewing on supersized 25x10 tiled display

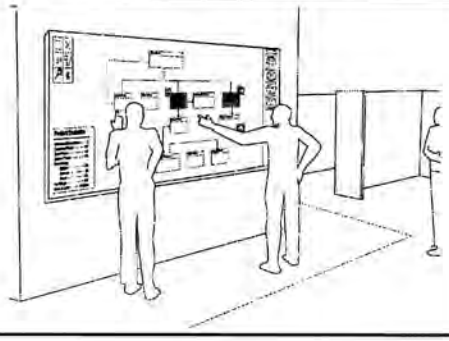
A Glimpse into the Future



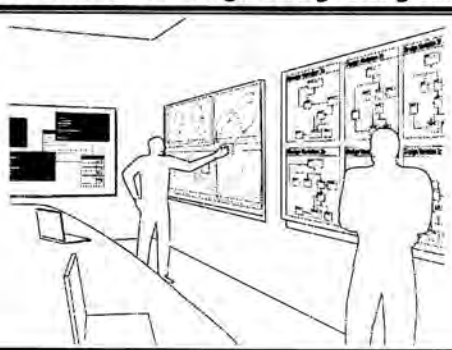
High-Tech Software Engineering: Awareness



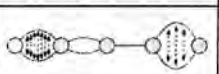
High-Tech Software Engineering: Dashboard



High-Tech Software Engineering: Design Room



Continuous Coordination

	Conceptual Visualization	Strengths	Weaknesses
Formal Coordination		Scalable; Insulation from other activities; Control; Group-first	Reconciliation problems; Insulation becomes isolation
Informal Coordination		Flexible; Promotes synergy; Raises awareness; User-first	Not scalable; Requires human-intermediation
Continuous Coordination		Scalable; Flexible; Self-coordination; Group- and user	Complex system building; full tradeoffs to be examined

Continuous Coordination: The Way Forward

Conclusions

- Continuous coordination is a principle that is critical to effective software development
- By understanding workspace activities, we can learn more about what really happens in parallel
- We have shown one example, configuration management, but virtually all software engineering needs to be based on the principle
- High-tech software engineering may be the key
  - information must be much easier to move around
  - workplaces must support the activities at hand
  - new software engineering tools must be built that leverage the high-tech equipment to *our* advantage

For More Information

- http://www.ics.uci.edu/~andre
- http://www.ics.uci.edu/~asarma/Palantir

Thank you

bren:school
information and computer sciences

Informatics
= computer science + people in context

招待講演

「化学と情報とコンピュータをつなぐ境界領域ウォーキング」

佐藤　寛子（国立情報学研究所）

講師紹介

佐藤　寛子（国立情報学研究所　知能システム研究系　助教授）
御茶ノ水女子大学理学部化学科博士課程出身.
専門は化学情報学，計算機化学.
http://research.nii.ac.jp/~hsatoh/

概要

現代では「歩き」は移動の主要手段ではなく，電車に乗るだけで目的地に到達できるため，出発点と終点の駅周辺さえ知っていれば日常生活に困ることはありません．しかし，こうした「点」としてしか知らない駅や見知らぬ駅を降りて，自分の足で歩いてつないでみようと考えてみると，思わぬ繋がりや障害物に気づくことがあります．

境界領域における研究もこれと似ています．つまり，個々の「点」として存在する専門領域とその間の視点に立ち問題に直面することで，領域間の繋がりや障壁の実体が見えてきます．それらを実質的につなげるには，自分の足で歩くこと，すなわち，実地に立って問題を捉えることが必要でしょう．

こうした視点から，化学と情報学，コンピュータ科学等に関わる境界領域である「化学情報学」や「情報化学」とよばれる分野について，各領域間の繋がりや問題点を，ソフトウェア工学との接点も絡めて捉えることが本講演の主題です．

「化学は経験の学問である」いわれます．つまり，化学情報・知識・経験は化学において不可欠なものです．また，化学構造式や分子模型などの視覚的要素も，直接見ることのできない物質や現象を分子レベルで解明する上で極めて重要です．これだけを見ても，コンピュータ科学との接点が多岐にわたることが推測されると思います．また，日本の化学系ソフトウェア市場は欧米製オブジェクトコードにより圧倒的な割合が占められ，ソフトウェアを創る技術や，高度な化学系ソフトウェアを構築するための基盤となるソースコードの蓄積が殆どなされてきていないという，深刻な問題を抱えています．欧米の追随でない独創的なコンピュータ化学研究の機動力を発揮し国際競争力を高めるには，根本をささえるソフトウェア基盤の早急な整備が必須であり，情報サービス産業との実質的な連携が望まれます．ここにもソフトウェア工学との密接な接点を見いだすことができるでしょう．

こうした話題について，最近の研究成果と合わせて紹介したいと思います．

化学と情報とコンピュータを つなぐ境界領域ウォーキング

国立情報学研究所
佐藤　寛子

1

自己紹介

■ 経 歴

お茶の水女子大学理学部化学科卒業
東レ株式会社
豊橋技術科学大学
理化学研究所 基礎科学特別研究員
科学技術振興事業団さきがけ研究21
国立情報学研究所 情報メディア研究系 助手
国立情報学研究所 知能システム研究系 助教授

2

自己紹介

■ 研究分野

化学情報学
情報化学
計算機化学
情報理論化学

3

自己紹介

■ 所属している学会

日本化学会
日本化学会情報化学部会
アメリカ化学会
日本農芸化学会
日本薬学会
日本コンピュータ化学会
人工知能学会

4

自己紹介

■ 研究テーマ

化学情報の表現法の開発
化学反応予測の研究
NMRスペクトル予測の研究
分子構造推定の研究
分子の可視化とデバイスに関する研究
化学教材の開発
上記ソフトウェアの開発

5

自己紹介

■ 著書・訳書

情報学シリーズ7
化学情報学
—化学反応の系図と反応予測—

佐藤 寛子 著

丸善株式会社
293 ページ
2003年
ISBN 4621071971

6

自己紹介

■ 著書・訳書

ケモインフォマティックス
予測と設計のための化学情報学

J. Gasteiger, T. Engel 編
船津公人, 佐藤寛子, 増井秀行 訳

丸善株式会社
638ページ
2005年
ISBN 4621075527

自己紹介

■ 受賞歴

情報化学討論会ポスター賞 (1996,2000)
有機合成化学協会 第一製薬研究企画賞 (1999)
科学技術振興事業団 第35回科学技術情報振興賞・論文賞 (2000)
日本化学会BCSJ賞 (2000)
日本薬学会 反応と合成の進歩シンポジウムポスター賞 (2001)

境界領域研究とウォーキング

「歩き」は移動の主要手段ではなくなった

出発点と終点の駅周辺さえ知っていれば
日常生活には不自由がない

境界領域研究とウォーキング

■ 山手線内の大きさ

山手線一周 約35km

東西：約 7 km

南北：約 14 km

面積：約 77 km^2

境界領域研究とウォーキング

■ 富山市に置き換えると...

山手線一周 約35km

東西：約 7 km

南北：約 14 km

面積：約 77 km^2

境界領域研究とウォーキング

■ 山手線と地下鉄の駅の数

山手線： 29駅

各駅間の平均距離：約 1.2 km

地下鉄： 107駅

1駅あたりの平均占有面積：
約 0.71 km^2

境界領域研究とウォーキング

■ 山手線と地下鉄の駅の数

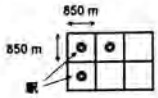
山手線： 29駅

各駅間の平均距離：約 1.2 km

地下鉄： 107駅

1駅あたりの平均占有面積：
約 0.71 km²

各駅間の平均距離： 約 850 m



13

境界領域研究とウォーキング

山手線内では...

約 1 km の駅間隔であっても，これらの間のつながりを知らないことが多い

しかし，点としてしか知らない駅や見知らぬ駅を降りて自分の足で歩いてみると，思わぬ繋がりや障害物に気づく事がある

境界領域研究との類似点

14

境界領域研究とウォーキング

点 —— 駅 —— 個々の専門領域

つなぐ —— 歩く —— 境界領域で研究する

歩くこと：

各専門領域と境界領域の実地に立って問題を捉えること

15

今日のテーマ

化学と情報学，コンピュータ科学等に関わる境界領域である「化学情報学」や「情報化学」とよばれる分野について，各領域間の繋がりや問題点をソフトウェア工学との接点も絡めて紹介する

16

今日のテーマ

1．分野や背景の違いによる誤解や理解の障壁

2．領域を越えた化学情報学研究の取り組み

3．化学ソフトウェアの現状と将来

17

分野や背景の違いによる誤解や理解の障壁

■ 各種メディアによる報道
— 新聞や雑誌，TVなど

取材される側
→ 「ある程度ギャップのある報道をされるのは覚悟しなければならない」

取材する側
→ あらかじめ作ったストーリに適う取材情報を都合よく貼り合わせて報道する

事実の断片が組み合わされて報道されているからといって，そのストーリー全体が真実であるとは限らない

18

今日のテーマ

1．分野や背景の違いによる誤解や理解の障壁

2．領域を越えた化学情報学研究の取り組み

3．化学ソフトウェアの現状と将来

25

領域を越えた化学情報学研究の取り組み

- 化学情報学という境界領域
- 研究の紹介

26

化学情報学とは

化学の諸分野の実践に適うかどうか？

基盤と応用

27

化学と情報

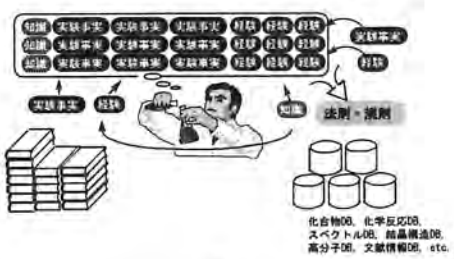
化学と情報は密接な関わりがある

- 化学の発展してきた歴史
  ex. 周期表の発見　1860年代
- 化学者の思考
- 化学におけるコンピュータ利用の始まり
  1960年代　分子情報に基づく分子構造推定

28

化学と情報

化学者はどのように研究を進めているか？



29

コンピュータ利用に対する化学者の一般的反応と姿勢

30

今日のテーマ

1．分野や背景の違いによる誤解や理解の障壁

2．領域を越えた化学情報学研究の取り組み

3．化学ソフトウェアの現状と将来

領域を越えた化学情報学研究の取り組み

- 化学情報学という境界領域
- 研究の紹介

化学情報学とは

化学の諸分野の実践に適うかどうか？

基盤と応用

化学と情報

化学と情報は密接な関わりがある

- 化学の発展してきた歴史
  ex. 周期表の発見　1860年代
- 化学者の思考
- 化学におけるコンピュータ利用の始まり
  1960年代　分子情報に基づく分子構造推定

化学と情報

化学者はどのように研究を進めているか？

コンピュータ利用に対する化学者の一般的反応と姿勢

我々の姿勢

もっと柔軟に

コンピュータは道具.
使う人間によって,
有効に活かせることもできれば,
役に立たないものにもなる.

- コンピュータにできることはあるか？
- 何をどう実行させるか？
- 出てきた結果をどう解釈するか？

人間の仕事

31

我々の姿勢

コンピュータはツールのひとつ

何を解決したいか？
何を知りたいか？

- どのような技術が必要か？
- どのような情報やデータが必要か？
- どうやって得るか？
- どう処理するか？ etc.

32

我々の姿勢

実践的な化学研究としての化学情報学研究

基盤から応用まで

33

研究の紹介 — データベースの品質管理

化学においてデータは要
実験や計算のデータを基盤として論理を積み上げる

情報化学において化学情報は最も重要な基盤
情報に基づいてモデル化や推論・予測を行う

34

研究の紹介 — データベースの品質管理

実践的な活用に適う種類，質，量を兼ね備えている

35

研究の紹介 — データベースの品質管理

化学反応データベースの品質管理

化学反応データベース
ISIS ChemInform
329件／約60万件

間違いのあった反応データ 件数：総数151

反応物・生成物等の構造	50
反応式	68
反応ステップ数	83
反応条件	31
反応部位	61
収率	18

Satoh, H., Nakata, T. *J. Comput. Chem. Japan*, 2, 87-94(2003)

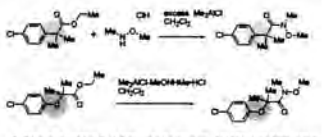
36

研究の紹介 — データベースの品質管理

化学反応データベースの品質管理

登録データ

正しいデータ



T. Shimizu, K. Osaki, T. Nakata, *Tetrahedron Lett.*, **38** (13), 2685-2688 (1997)

結果例ー１：平面構造式が間違っている

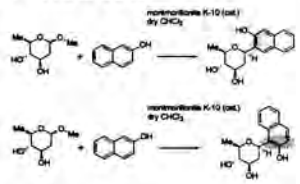
37

研究の紹介 — データベースの品質管理

化学反応データベースの品質管理

登録データ

正しいデータ



K. Tenkima, Y. Ushiki, G. Matsuo, S Matsumura, *Tetrahedron Lett.*, **38**, 7375-7378 (1997)

結果例２：置換位置が間違っている

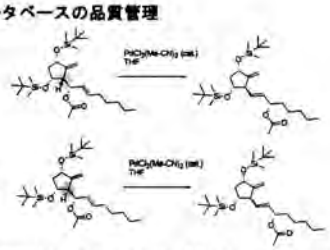
38

研究の紹介 — データベースの品質管理

化学反応データベースの品質管理

登録データ

正しいデータ



M. Nakazawa, Y. Sakamoto, T. Takahashi, K. Tsuneoka, K. Ishikawa, T. Nakai, *Tetrahedron Lett.*, **34** (37), 5923-5926 (1993)

結果例３：立体配置が間違っている

39

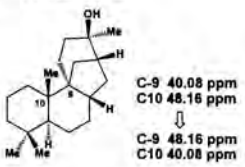
研究の紹介 — データベースの品質管理

NMRデータベースの品質管理

40

研究の紹介 — データベースの品質管理

NMRデータベースの品質管理



C-9 40.08 ppm
C10 48.16 ppm
⇩
C-9 48.16 ppm
C10 40.08 ppm

Aphidicolan-16β-ol

Tetrahedron, **55**, 7541-7554 (1999) H. Oikawa et al.
Dr. H. Oikawa, *personal communication*

結果例—１　帰属が間違っている

41

研究の紹介 — 化学反応予測

化学反応予測とは？

反応物と反応条件とから何がどれだけ生成するかを予測すること

反応物，反応条件 ⟶ ？

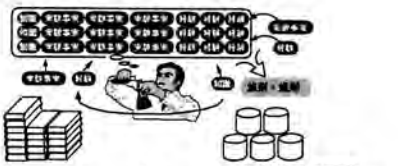
42

研究の紹介 — 化学反応予測

化学反応を一般的に記述できる理論は確立されていない

化学反応の多種多様性

研究の紹介 — 化学反応予測



＋

コンピュータの特色
（網羅性，高速性，正確さ，大容量の記憶領域）

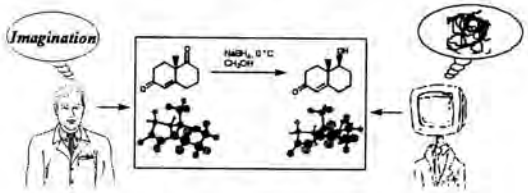
化合物DB，化学反応DB，
スペクトルDB，結晶構造DB，
高分子DB，文献情報DB，etc.

↓

反応予測のための法則や規則を導き出す新しいアプローチ？

研究の紹介 — 化学反応予測

コンピュータに適した反応表現とは？



研究の紹介 — 化学反応予測

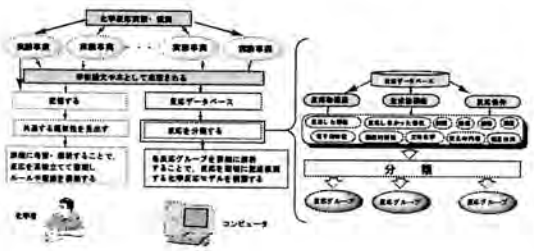
化学反応の経路を決める種々の因子 → 多次元空間におけるデータ

化学反応 → 多次元データの非線形なつながり

↓

コンピュータに適した新しい反応表現法？

研究の紹介 — 化学反応予測



研究の紹介 — 化学反応予測

試薬の持ち得る機能の可能性の程度を予測する

研究の紹介 — 化学反応予測

試薬機能予測モデルの構築

FRAUシステムによる分子特性の数値化

↓

FRAU特性値に基づく分類とモデル化

↓

予測と検証，モデルへのフィードバック

研究の紹介 — 化学反応予測

1. FRAUシステムによる分子特性の数値化

静電的相互作用に基づく特性値
立体的相互作用に基づく特性値
表面積に基づく特性値

研究の紹介 — 化学反応予測

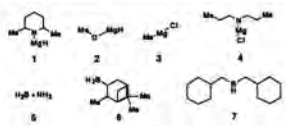
2. FRAU特性値に基づく分類とモデル化

a boranes
b borane complexes
c hydrides
d reducing agents containing atom
e reducing agents containing atom
f bases
g Grignard reagents

研究の紹介 — 化学反応予測

3. 予測・検証・フィードバック

	1	2	3	4	5	6	7
アルデヒドからアルコールへの還元	0.0	0.1	0.0	0.2	1.0	0.8	0.8
ケトンからアルコールへの還元	1.0	1.0	0.0	0.1	1.0	1.0	1.0
カルボン酸からアルコールへの還元	0.0	0.1	0.0	0.0	0.0	0.8	0.5
エステルからアルデヒドへの還元	0.0	0.1	0.0	0.0	0.0	0.1	0.0
エステルからアルコールへの還元	0.0	0.1	0.0	0.0	0.0	0.1	0.4
エポキシ環の還元	0.0	0.1	0.0	0.0	0.0	0.1	0.6
ヒドロホウ素化	0.0	0.0	0.0	0.0	0.0	1.0	1.0
アルドール縮合（塩基として働く）	0.0	0.0	0.3	0.9	0.0	0.0	0.0
アルキル化	0.0	0.0	1.0	0.4	0.1	0.0	0.0



1 2 3 4 5 6 7

研究の紹介 — NMRスペクトル予測

NMR: Nuclear Magnetic Resonance
核磁気共鳴

分子構造を決定するための分光学の
主力手段

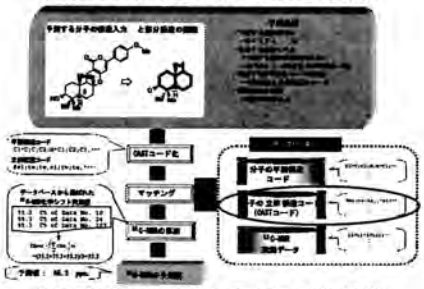
研究の紹介 — NMRスペクトル予測

目　的

実践的な分子構造解析に活用できる
予測精度をもつソフトウェアを開発する

研究の紹介 — NMRスペクトル予測

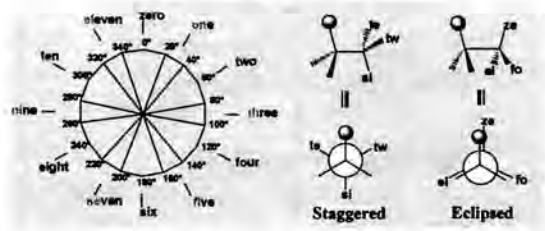
^{13}C NMR高精度予測システム CAST/CNMR



H. Satoh, H. Koshino, J. Uzawa, T. Nakata, *Tetrahedron*, 59, 4539-4547(2003)

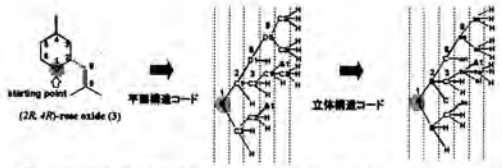
研究の紹介 — NMRスペクトル予測

分子構造コード化法CAST



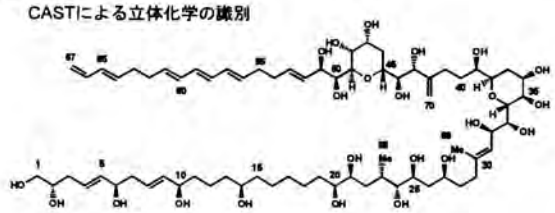
研究の紹介 — NMRスペクトル予測

分子構造コード化法CAST



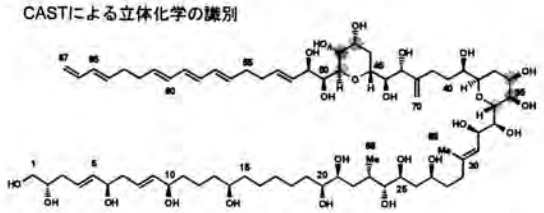
研究の紹介 — NMRスペクトル予測

CASTによる立体化学の識別



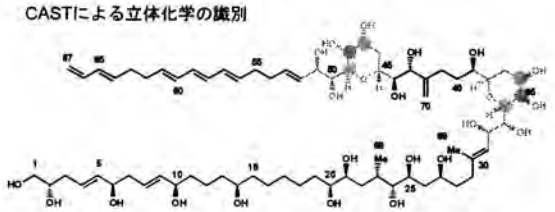
研究の紹介 — NMRスペクトル予測

CASTによる立体化学の識別



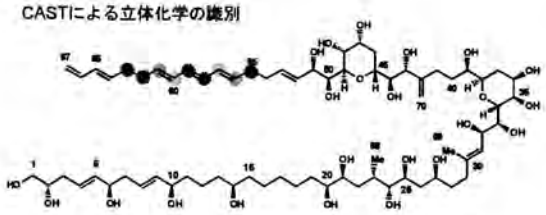
研究の紹介 — NMRスペクトル予測

CASTによる立体化学の識別



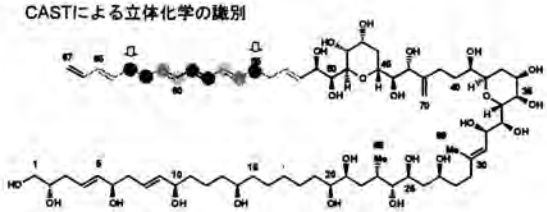
研究の紹介 — NMRスペクトル予測

CASTによる立体化学の識別



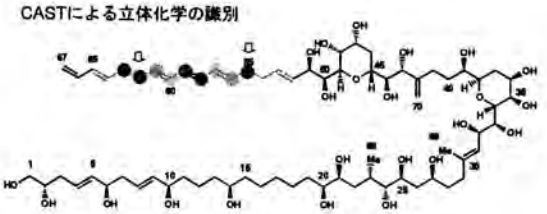
研究の紹介 — NMRスペクトル予測

CASTによる立体化学の識別

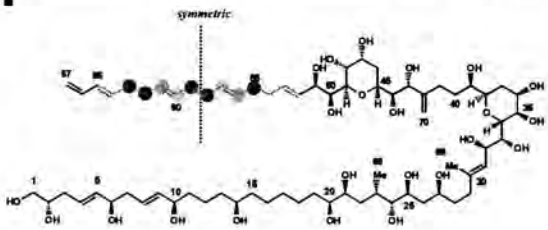


研究の紹介 — NMRスペクトル予測

CASTによる立体化学の識別

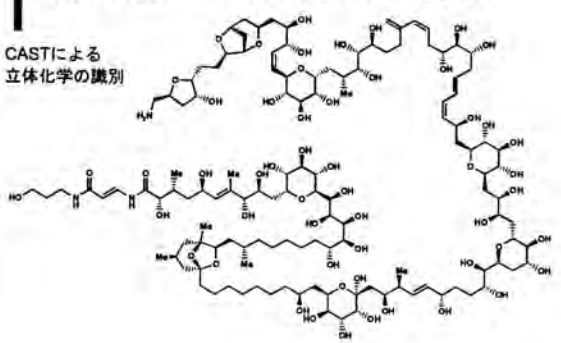


研究の紹介 — NMRスペクトル予測



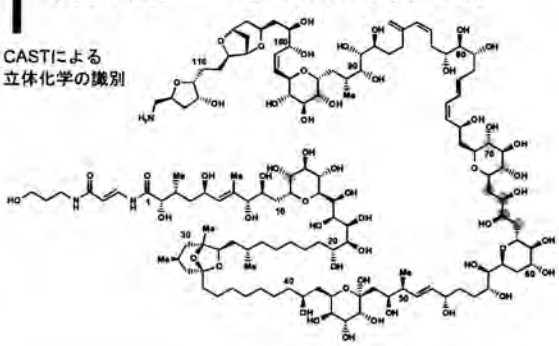
研究の紹介 — NMRスペクトル予測

CASTによる
立体化学の識別



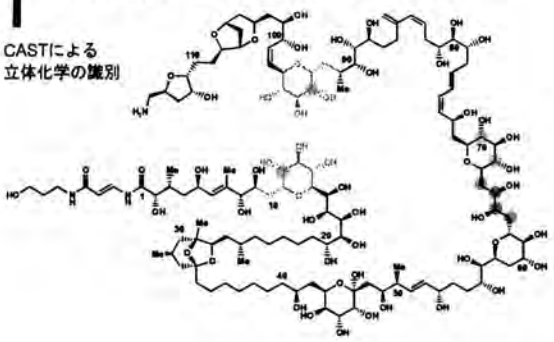
研究の紹介 — NMRスペクトル予測

CASTによる
立体化学の識別



研究の紹介 — NMRスペクトル予測

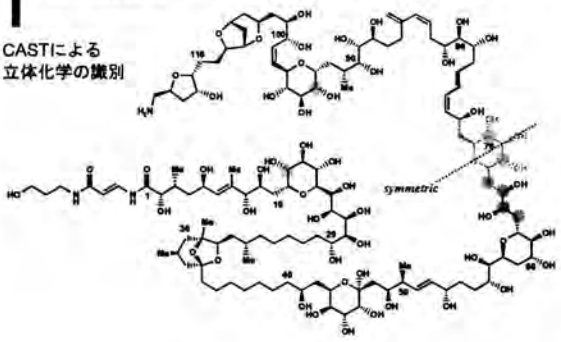
CASTによる
立体化学の識別



67

研究の紹介 — NMRスペクトル予測

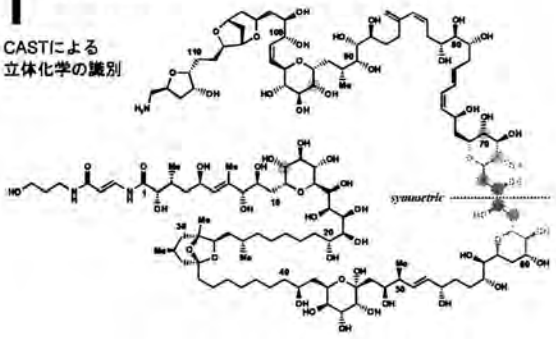
CASTによる
立体化学の識別



68

研究の紹介 — NMRスペクトル予測

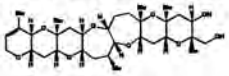
CASTによる
立体化学の識別



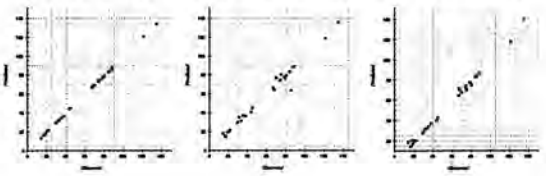
69

研究の紹介 — NMRスペクトル予測

市販システムとの予測精度比較



CAST/CNMR　SpecInfo (Germany)　ACD (Canada)



70

研究の紹介 — 分子情報の可視化

化学は視覚的要素の強い学問分野

化学構造式、立体化学表記、分子軌道図etc.
種々のコンピュータグラフィックス・モデリングソフトウェア
↳ システムの見栄え

- 可視化ソフトウェアはアカデミックな化学研究と認められにくい
- ソースコードが開示されていないことが多い→複数の人間が同じ努力
- オープンソース化されているソフトウェアは生命科学系と比較して圧倒的に少ない

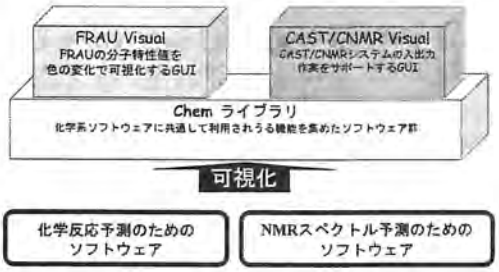
71

研究の紹介 — 分子情報の可視化

分子情報可視化のための
グラフィックライブラリ構築

72

研究の紹介 — 分子情報の可視化



今日のテーマ

1．分野や背景の違いによる誤解や理解の障壁

2．領域を越えた化学情報学研究の取り組み

☞3．化学ソフトウェアの現状と将来

化学ソフトウェアの種類

文献データベース	➡ 最新動向のチェック 関連分野のレビュー
分子構造描画・モデリングシステム	➡ 文書作成， ソフトウェアの入出力表示
分子設計・材料設計	➡ 有効な機能をもつ分子や材料の 設計（ex. 創薬，試薬）
化学合成設計・反応予測	➡ 合成ルートを設計し， 合成できるかを予測する．
分子構造推定・分光スペクトル予測	➡ 合成・単離したものの 分子構造を決定する
測定機器の操作用ソフトウェア	➡ 機器とのインタフェース

化学ソフトウェアの現状

■ 化学ソフトウェアの市場—1

一部の企業や団体が独占

ソースコードは非公開

化学ソフトウェアの現状

■ 化学ソフトウェアの市場—2

欧米支配

日本は欧米のオブジェクトコードに依存

化学ソフトウェアの現状

■ 日本の化学ソフトウェアの現状

化学ソフトウェアの基盤技術・人材と
ソースコードが枯渇している

ソフトウェアを使う技術はあっても
創る技術が蓄積されていない

根のない切り花を買ってきているのと同じ．
花を育て増やすことも，改良することもできない．

ソースコードの蓄積がなされていないことは，
将来の発展に不可欠な基盤が築かれていない
ことを意味する

化学ソフトウェアの将来

化学基盤ソフトウェアのオープンソース化が必要

- 共有財産としての知的基盤の構築
- 人材育成

ソフトウェアの発展には競争原理も必要

ソースコードを非公開とし権利と利益を
独占する部分と共通技術とを区別することが必要

まとめ

- 関連領域が実質的に融合された境界領域研究には
  自分の足で実地に立って物事を捉える努力が必要である
- 実践的な化学研究としての化学情報学研究には
  実践を鑑みた基盤から応用までの研究が不可欠である
- 化学ソフトウェアの発展のためには基盤技術を
  オープンにし共有・活用することが必要である

SEA Forum Special

ソフトウェア工学教育の新しい試み

6月6日(月曜) １６：００～１８：００

@ 東京大学（駒場キャンパス）先端科学技術研究センター ４号館 ２階講堂

ソフトウェア・シンポジウム 2005 (6/8-10 @ 富山国際会議場) の基調講演者である Andre van der Hoek 先生（UCIrvine）が，富山への途中，東京に立ち寄られる機会に，先生を講師にお迎えして表記の特別フォーラムを企画しました．

van der Hoek 先生 は，Configuration Management や Software Architecture あるいは Product Line Approach など，ソフトウェア工学の多彩な分野の研究を手がけておられる新進気鋭の研究者ですが，大学におけるソフトウェア工学の教育にも深い関心を持ち，いくつかの実験的プロジェクトを試行してすぐれた成果をあげておられます．

今回のフォーラムでは，そうした研究開発の成果をご報告いただき，実践的なソフトウェア技術者教育のあるべき姿を討論したいと考えています．

なお，van der Hoek 先生の研究業績や，教育関連プロジェクトの概要については，次の Web Page をご覧ください．

個人 Web Page： http://www.ics.uci.edu/~andre/
(Publication のところに，これまでに発表された教育関連の論文が10数編載っています)

SimSE (Game-based Software Engineering Simulation Environment)
http://www.ics.uci.edu/~emilyo/SimSE/

Problems and Programmers (Software Engineering Card Game)
http://www.problemsandprogrammers.com

The Changing Face of Software Engineering Education

André van der Hoek
Donald Bren School of Information and Computer Sciences
Department of Informatics
andre@ics.uci.edu

June 2005

Defining Software Engineering

- "*The establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works on real machines.*" [Bauer]
- "*Software engineering is that form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems.*" [CMU/SEI-90-TR-003]
- "*The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.*" [IEEE]

Teaching Software Engineering

- "*The establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works on real machines.*" [Bauer]
- "*Software engineering is that form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems.*" [CMU/SEI-90-TR-003]
- "*The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.*" [IEEE]

Defining/Teaching Software Engineering

- "*A discipline that deals with the building of software systems which are so large that they are built by a team or teams of engineers.*" [Ghezzi, Jazayeri, Mandrioli]
- "*Multi-person construction of multi-version software.*" [Parnas]
- "*A discipline whose aim is the production of fault-free software, delivered on-time and within budget, that satisfies the user's needs. Furthermore, the software must be easy to modify when the user's needs change.*" [Schach]
- "*It's where you get to design big stuff and be creative.*" [Taylor]

Defining/Teaching Software Engineering

- "*Difficult*" [van der Hoek]

How Difficult?

- SIGCSE (Technical Symposium on Computer Science Education)
- CSEE&T (Conference on Software Engineering Education & Training)
- ITiCSE (Conference on Innovation and Technology in Computer Science Education)
- FIE (Frontiers in Education Conference)
- IASTED CATE (IASTED International Conference on Computers and Advanced Technology in Education)
- ICSE education track (International Conference on Software Engineering)
- ...

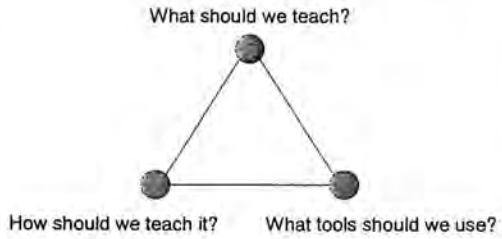
Some Example Innovations

- An entire course on software inspection
- Use of outside customers for class project
- "20 dirty tricks"
  - give additional tasks
  - change deadlines
  - change team / working procedures
  - ...
- Application of personal software process
- Use of extreme programming
- ...

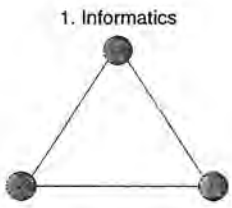
Impact

- A significant portion of the papers follow "the whim" of industry
  - personal software process
  - extreme programming
  - ...
  - but how much will last?
    - pair programming is perhaps an exception
- Many approaches can, realistically, not be replicated
  - an entire course on software inspection
- Most improvements, though certainly helpful, are spot solutions, addressing one small aspect of the larger problem
  - 20 "dirty tricks"

Reformulating the Problem



Reformulating the Problem



Motivation

- Software engineering always takes place in context
  - software is never the objective, it is always the use of the software that is the objective
- One or a few courses on software engineering are simply not sufficient
  - too much material must be taught
- We cannot focus on just basic knowledge and basic skills
  - the focus too often is on notations and deliverables only

Approach

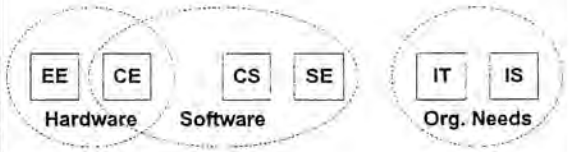
- Focus on software engineering in context
  - move to software *and* information
  - move to development *and* design
  - move to technical *and* social
  - move to synthesis *and analysis*
- Create a new, four-year degree program
  - Informatics
- Seek a balance among advanced knowledge, extensive practical skills, and creative intuition

Informatics = Computers + People

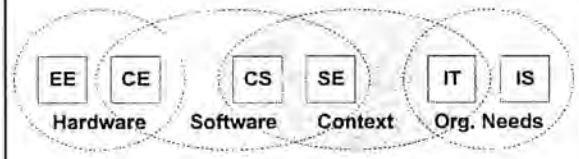
- Interdisciplinary study of the design, application, use, and impact of information technology
- Broadly speaking: software engineering *in context*
  - inherently inter-disciplinary
  - focus more on designing real-world solutions, less on building infrastructure
- A different alternative for software engineering education
  1. mathematics and engineering orientation
  2. SE 2004 recommended CS core with specialized software courses
  3. *context-based software engineering (UC Irvine)*

ACM Computing Curricula 2004 – Pre 1990

ACM Computing Curricula 2004 – Post 1990



The Informatics Focus



A Software Engineering Perspective

- A primary focus is software and information systems and how we can develop them properly
- Students are trained to have a "tool box"
  - how to design
    - much more than the traditional notation only approach
  - how to program
  - overall lifecycle
  - role of methods and tools
  - ...
- Students are trained to be creative, work in teams, and generally develop solutions, not just programs
- We cover, on top of traditional topics:
  - "little languages"
  - design techniques
  - design patterns
  - real-time issues
  - distributed and decentralized systems
  - design-to-code mappings
  - maintenance
  - project management
  - ...

A Context Perspective

- A primary focus is context, why we are developing software and information systems and where such systems are deployed
- Students are trained to have a "tool box"
  - ethnographic studies
  - user studies
  - design techniques
  - ethics, privacy, security
  - ...
- Students are trained to know the technology available to realize their solutions
- We cover, on top of software engineering topics:
  - user interfaces
  - human-computer interaction
  - social analysis of computerization
  - organizational issues
  - project management
  - collaboration
  - computer-supported cooperative work
  - information retrieval
  - information visualization
  - ...

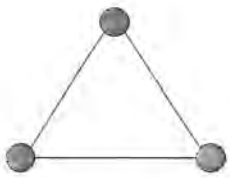
Curriculum

Fall Year 1	Winter Year 1	Spring Year 1
Informatics Core	Informatics Core	Informatics Core
		Informatics Research Topics
Writing	Writing	Writing
Critical Reasoning	Discrete Mathematics	Fundamental data structures
Fall Year 2	Winter Year 2	Spring Year 2
Statistics	Human-Computer Interaction	Project in HCI and User Interfaces
Concepts in Programming Lang. I	Concepts in Programming Lang II	Software Design I
Software Methods & Tools	Requirements Analysis & Engr.	SW Specification & Quality Engr.
Breadth	Breadth	Breadth
Fall Year 3	Winter Year 3	Spring Year 3
Social Analysis of Computerization	Organizational Information Systems	Proj in Social & Org Impacts of Comp
Software Design II	SW Arch, Dist Syst, & Interoperability	File and Database Management
Breadth	Breadth	Breadth
Breadth / Elective	Breadth / Elective	Breadth / Elective
Fall Year 4	Winter Year 4	Spring Year 4
Senior Design Project	Senior Design Project	Senior Design Project
Project Management	Computer-Supported Coop Work	Breadth / Elective
Proj. in File and Database Mgmt	Information Retrieval	Information Visualization
Breadth / Elective	Breadth / Elective	Breadth / Elective

Observations

- Multiple course sequences
- Incremental delivery of materials
- Equal attention to software engineering and context
- Integration through end-of-year project courses
- Ample room for basic and advanced topics, basic and advanced training, and slowly-but-surely instilling an intuition in students

Reformulating the Problem



2. Learning theories

Motivation

- A typical course in software engineering consists of theory and a course project
  - theory: life cycles, individual phases, documents, properties, notations, some principles, ...
  - project: usually, an exercise of the waterfall life cycle model with a focus on deliverables
- Designed based on *what* to teach
- Not designed based on *how* to teach
  - what are good educational approaches?
  - what approaches are better for which subjects?
  - can we build better courses with better educational delivery methods?

Approach

- A principled approach to course design and course delivery based on learning theories
- Learning theories describe *how* people learn
  - learning by doing
  - situated learning
  - Keller's ARCS motivation theory
  - anchored instruction
  - discovery learning
  - learning through failure
  - learning through dialogue
  - learning through reflection
  - elaboration
  - ...

Learning by Doing

- People learn best by actively *doing* something, rather than simply hearing / reading about it
- The learner should be given ample opportunity to:
  - practice
  - analyze
  - reflect
  - synthesize
- Common in course projects, but usually focuses on synthesis, the one-time creation of deliverables

Situated Learning

- Knowledge is *situated* – a product of the activity, context, and culture in which it is developed and used
- The learning environment should resemble the environment in which the knowledge will be practiced as closely as possible
- Highly uncommon in course projects, only when students are placed at a real customer's site

Anchored Instruction

- All learning / teaching activities should be centered around an "anchor" – a realistic case study, situation, or problem
- Knowledge is learned best through exploration of anchors
- Presentation of general concepts and theories kept to a minimum
- Highly uncommon

Learning through Dialogue

- Knowledge is best learned when dialogue sessions with the instructor are incorporated
- Instructor should encourage reflection, assess aptitude and learning style, and tailor instruction accordingly
- Exists somewhat, but certainly not widely used

Example Approaches versus Learning Theories

	Learning by Dialogue et. al.	Situated Learning et. al.	Keller's ARCS	Anchored Instruction	Discovery Learning	Learning Through Failure	Learning Through Dialogue	Aptitude-Treatment Interaction	Multiple Intelligences	Learning through Reflection	Rhizomics	Lateral Thinking
Industrial Partnership – Real Project	X	X	X				D/P			D/P		
Team Composition – Long-Term Teams	X	X					P			P	X	P
Open-Endedness – Requirements	X	X	X		X	X	P			P		
Open-Endedness – Process	X	X	X		X	X	P			P		
Practice-Driven	X			X	X	X	D/P	P	?	D/P	P	
Sabotage	X	X				X	P			P	P	
Re-enacting Failures	X	X				X	P			P	P	
Simulation – Industrial	X	X	X	P	X	X	P	P	?	P	D/P	P
Simulation – Game-Based	X	X	X	P	X	X	D/P	P	?	D/P	P	P
Simulation – Group Process	X	X	X	P	X	X	D/P	P	?	D/P	P	D/P

Example Approaches versus Learning Theories

Missing Piece	X											
Industrial Partnership – Modify Real Software	X	X					P			P	P	
Industrial Partnership – Industrial Advisor	X	X					P			P	P	
Industrial Partnership – Industrial Mentor	X	X					P			P	P	
Industrial Partnership – Case Study	X	X					P			P	P	
Maintenance/Evolution – Multi-semester	X	X					P			P	P	
Maintenance/Evolution – Single-semester	X	X					P			P	P	
Team Composition – Large Teams	X	X					P			P	P	
Team Composition – Different C.S. Classes	X	X					P			P	P	
Team Composition – Different Majors	X	X					P			P	P	
Team Composition – Different Universities	X	X					P			P	P	
Team Composition – Different Countries	X	X					P			P	P	
Team Composition – Team Structure	X	X					P			P	P	
Non-Technical Skills	X	X					P			P	P	

Frequency versus Learning Theories

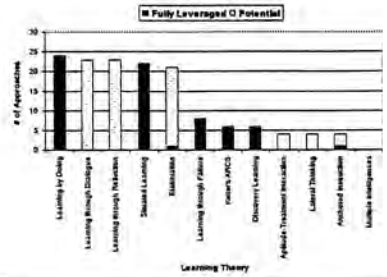
Realism	53	Simulation	8	Missing Piece	48
Industrial Partnerships	16	Industrial	2	Formality	5
- Modify real software	1	Game-Based	4	- Formal methods	2
- Industrial advisor	1	Group Process	2	- Engineering	3
- Industrial mentor/collectors	2			Process (Specific)	32
- Case study	5			- PSP	14
- Real project / customer	7			- TSP	2
Maintenance/Evolution	9			- RUP	2
- Multi-semester	4			- XP	2
- Single-semester	5			Process (General)	8
Team Composition	13			- Process engineering	2
- Long-term teams	1			- Project management	2
- Large teams	3			Parts of Process	3
- Different C.S. classes	1			- Scenario-based req. eng.	1
- Different majors	2			- Code reviews	1
- Different universities	2			- Usability testing	1
- Different countries	1			Types of Software Eng.	8
- Team structure	2			- Measurement/Evaluation	3
Non-Technical Skills	2			- Component-based SE	2
Open-Endedness	7			- Real-time SE	3
- Requirements	2			Non-Technical Skills	7
- Process	3			- Social/Ingratiatory skills	2
Practice-Driven	3			- Interact w/ stakeholders	1
Sabotage	2			- HCI	3
Project Failures	1			- Business aspects	1

Observation #1

The most popular approaches utilize the fewest learning theories

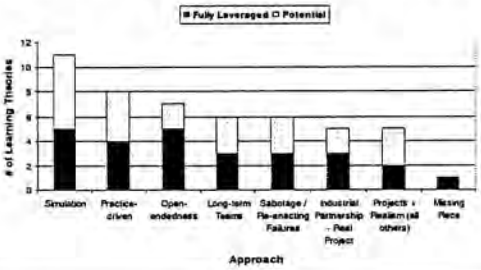
Observation #2

A number of learning theories are overlooked and under-utilized

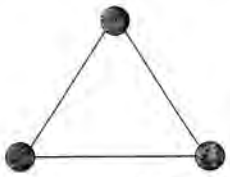


Observation #3

The educational potential of most approaches is not being maximized



Reformulating the Problem



3. Novel educational tools

Motivation

- A typical course in software engineering consists of theory and a course project
  - theory: life cycles, individual phases, documents, properties, notations, some principles, ...
  - project: usually, an exercise of the waterfall life cycle model with a focus on deliverables
- But the tools we have available to support the educational activities are poor
  - illustration of theory?
  - practice theory?
  - specific educational support?

Approach

- Investigate new tools specifically developed to support software engineering education
- Provide course modules along with the tools
- Build the tools and course modules with learning theories in mind
- Specifically created to reach "intuition"
- Three examples today
  - Problems and Programmers – an educational card game geared towards teaching software process issues
  - SimSE – an educational software process simulation game environment
  - EASEL – an educational software design environment

Problems and Programmers

- Two-player, physical card game that simulates the software process
  - focuses on process issues rather than deliverables
  - simulates a large project in a short time
  - represents a safe environment
  - can be played repeatedly
  - supports collaborative learning

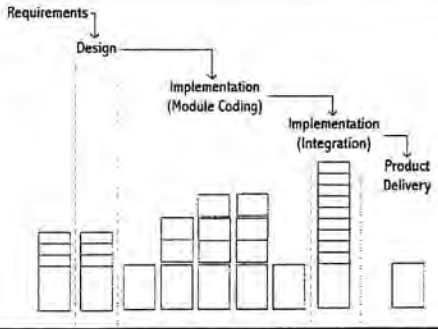
Game Objectives

- Advocate proper use of software engineering techniques
- Illustrate both generic and specific lessons concerning the software process
- Provide a student with clear feedback concerning their decisions
- Game play should be easy and comparatively quick
- Encourage interaction among students

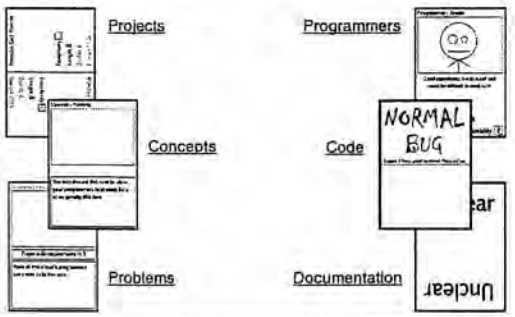
Overall Mechanics

- Two players "race" to complete the same project
- Software process is simulated from initial requirements to eventual delivery
  - waterfall with the option of going back (with a penalty)
- Different kinds of learning
  - layout and nature of play
  - game cards
  - interaction with opponent

Game Layout

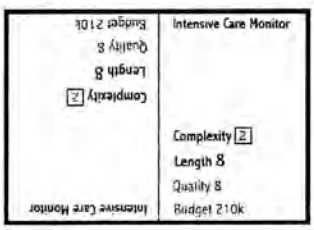


Card Types



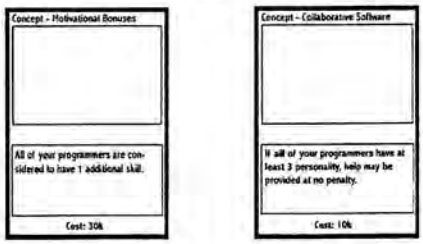
Project Cards

Different projects require different approaches



Concept Cards

Provide a positive effect, often for a cost



Problem Cards

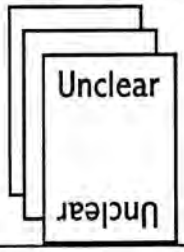
Problems are played on an opponent.
Each has a *condition* and an *effect.*

Programmer Cards

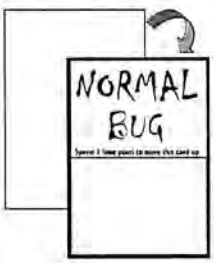
Skill, salary, and personality must be considered when programmers code, inspect, and debug

Documentation Cards

Represent time spent on requirements and design, and can be clear or unclear



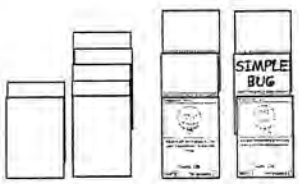
Code Cards

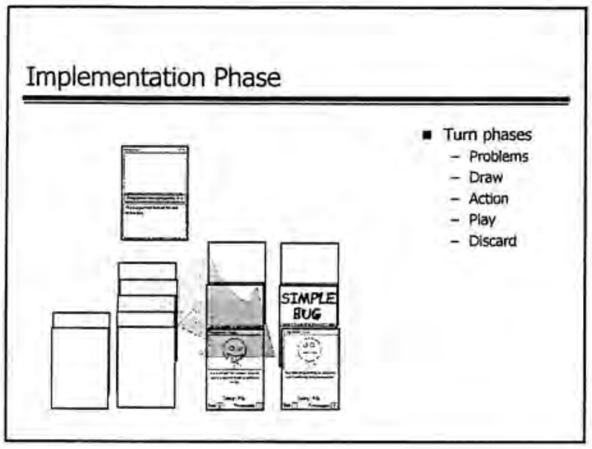


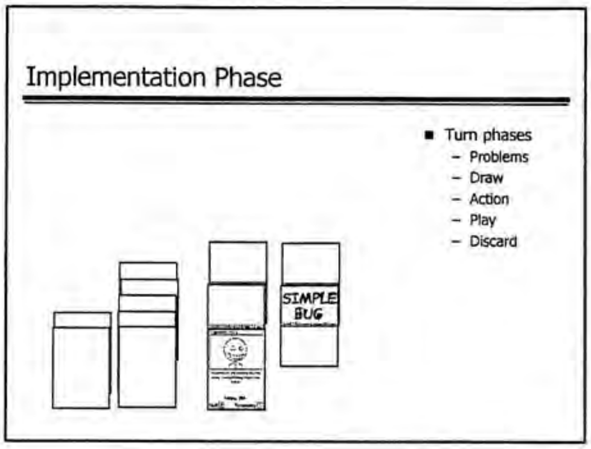
- Represent programming progress
- Based on skill, a programmer can:
  - create good code (cost = project complexity)
  - create rush code (cost = half project complexity)
  - inspect completed code (cost = 1 point)
  - fix bugs (cost = 1 point)

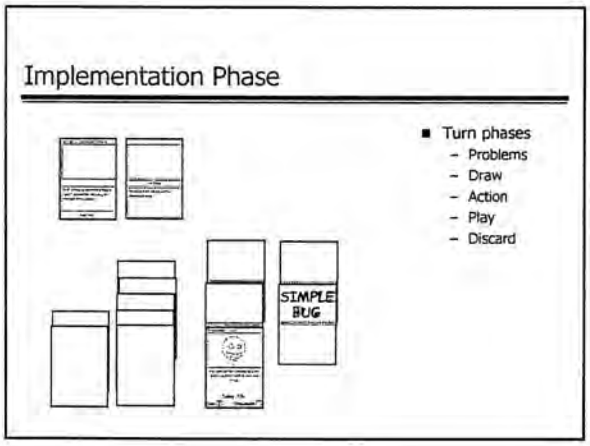
An Example – Implementation Phase

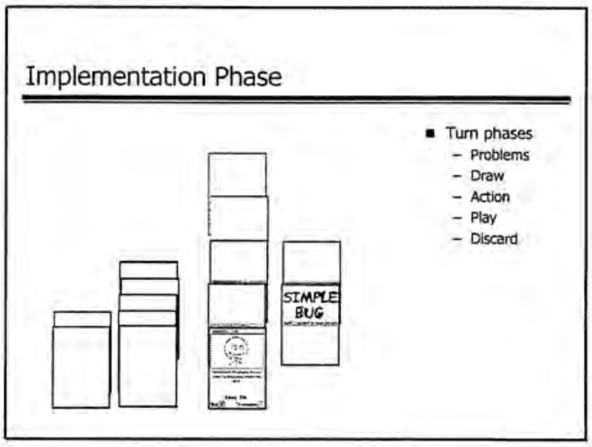
- Turn phases
  - Problems
  - Draw
  - Action
  - Play
  - Discard

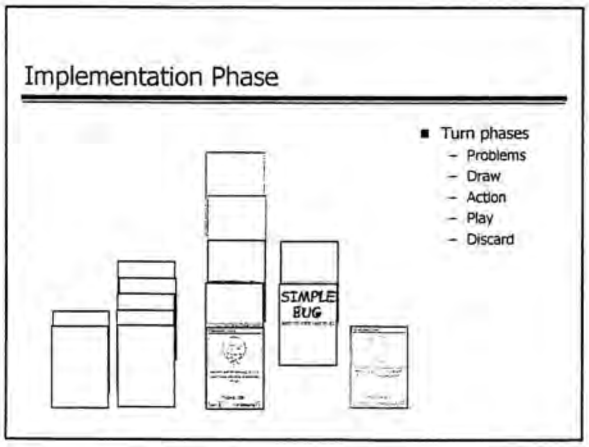


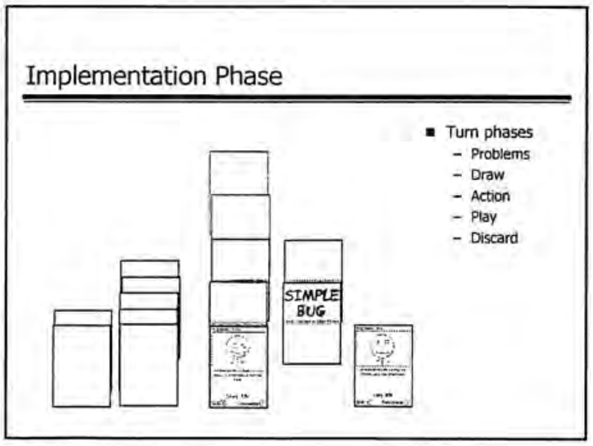

Implementation Phase
Turn phases
– Problems
– Draw
– Action
– Play
– Discard
SIMPLE BUG


Implementation Phase
Turn phases
– Problems
– Draw
– Action
– Play
– Discard
SIMPLE BUG


Implementation Phase
Turn phases
– Problems
– Draw
– Action
– Play
– Discard
SIMPLE BUG

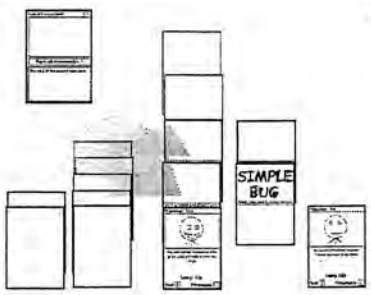

Implementation Phase
Turn phases
– Problems
– Draw
– Action
– Play
– Discard
SIMPLE BUG


Implementation Phase
Turn phases
– Problems
– Draw
– Action
– Play
– Discard
SIMPLE BUG


Implementation Phase
Turn phases
– Problems
– Draw
– Action
– Play
– Discard
SIMPLE BUG

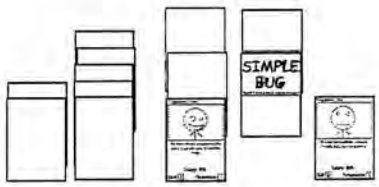
Implementation Phase

- Turn phases
  - Problems
  - Draw
  - Action
  - Play
  - Discard



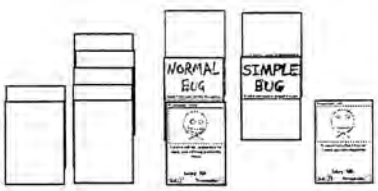
Implementation Phase

- Turn phases
  - Problems
  - Draw
  - Action
  - Play
  - Discard



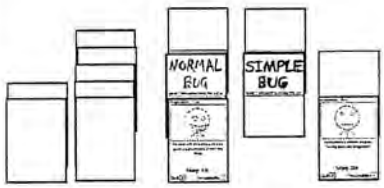
Implementation Phase

- Turn phases
  - Problems
  - Draw
  - Action
  - Play
  - Discard

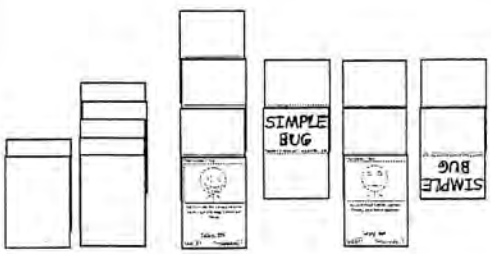


Implementation Phase

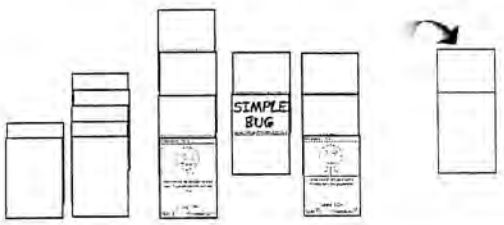
- Turn phases
  - Problems
  - Draw
  - Action
  - Play
  - Discard

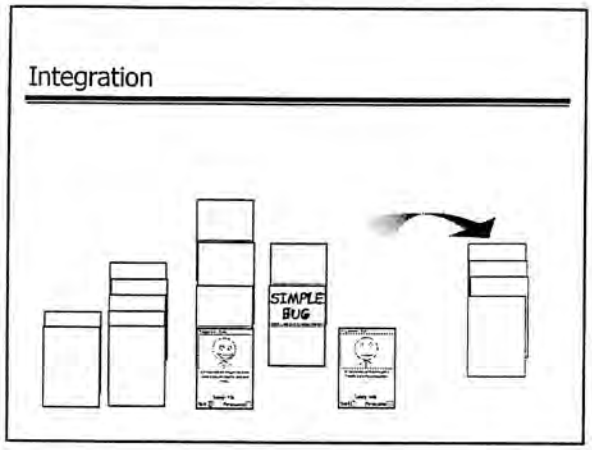


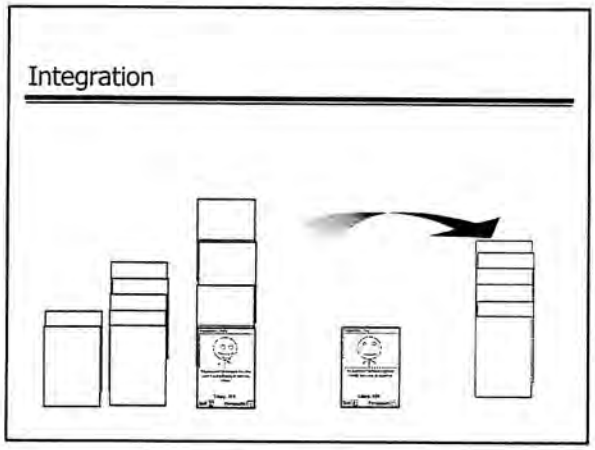
Fast Forward

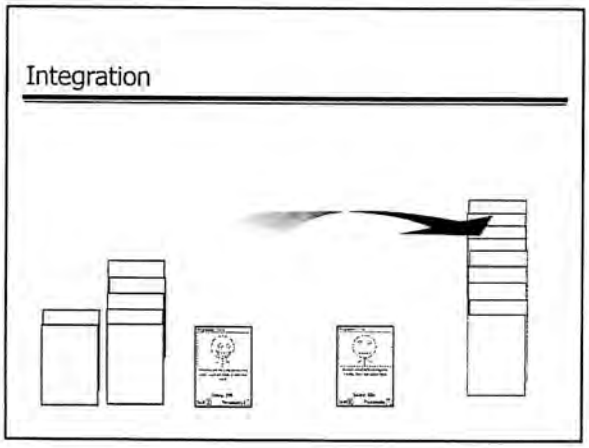


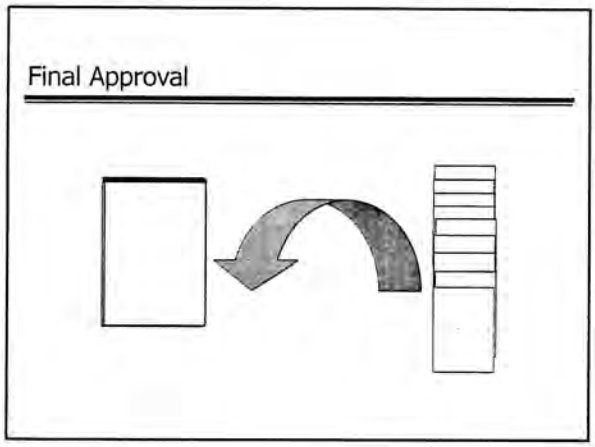
Integration

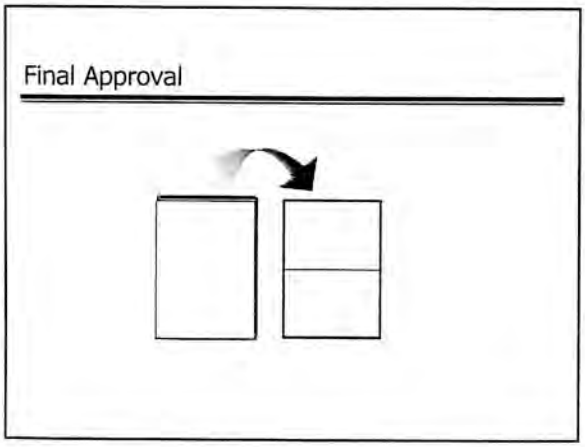


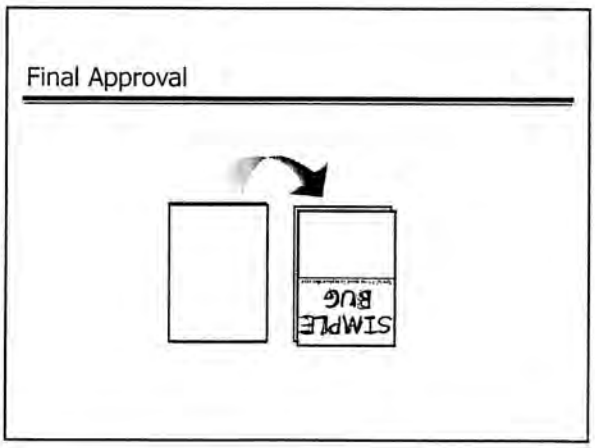

Integration
SIMPLE BUG


Integration

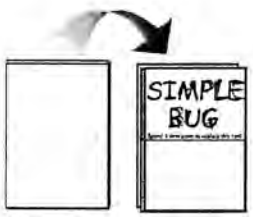

Integration


Final Approval


Final Approval


Final Approval
SIMPLE BUG

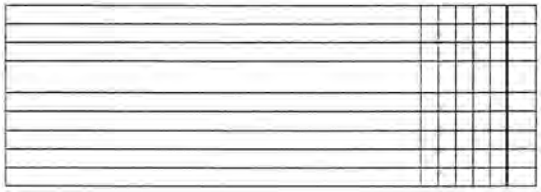
Final Approval



Experiment Setup

- 28 test subjects played the game
  - each had taken at least an introductory software engineering course
- In 2 ½ hours, subjects
  - learned the game
  - played it once or twice
  - filled out a questionnaire on the game

Numerical Results



Written Feedback - Positive

- *"You need to put the time into earlier phases (design) or else it will come back to get you."*
- *"It reinforces the ideas taught in ICS 52 with a fun way to learn it."*
- *"The game illustrates in 1 hour or 2 almost half of the material in ICS 52."*
- *"I think I learned that having many programmers = more time in integrations."*
- *"Tells me why it is important to create quality code."*

Written Feedback - Negative

- *"Requirements and design are boring."*
- *"Add different life cycles."*
- *"It was unclear the amount of time should be spent on requirements and design."*

A Repeat Experiment

- Approach at the University of Queensland
  - play Problems and Programmers
  - write a requirements specification for an online version of Problems and Programmers
  - reflect on the Problems and Programmers experience
- Very similar feedback
  - considered very helpful
  - reflective exercise helped a lot
  - early phases of the card game are boring
  - no choice of life cycle

Current Work

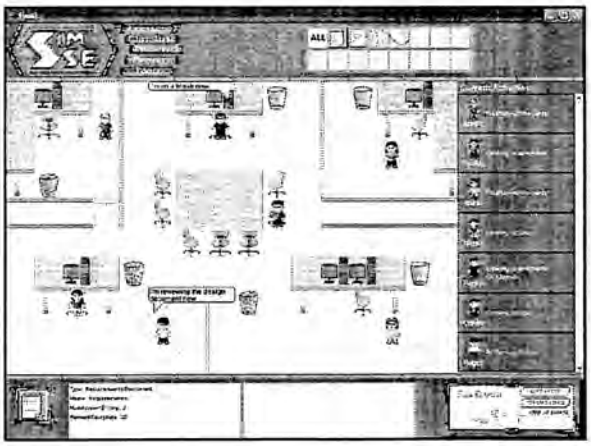
- Rethinking the game entirely
- Main objectives
  - make requirements and design more interesting
  - add additional lessons and concepts
  - maintenance
  - different life cycles
- Cards downloadable (but must be produced locally)

SimSE

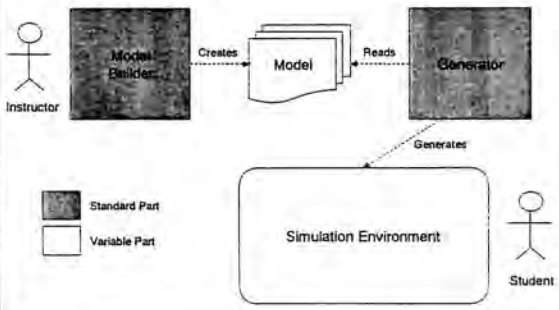
- Single-player computer game that simulates the software process
  - focuses on process issues rather than deliverables
  - simulates a large project in a short time
  - represents a safe environment
  - can be played repeatedly
  - supports reflective learning

Overall Mechanics

- SimSE is an interactive, graphical software engineering simulation game
- The student
  - takes on the role of project manager
  - must manage and drive the process by:
    - assigning tasks to employees
    - hiring/firing
    - monitoring progress
    - purchasing/using tools
  - receives a score at the end



Architecture



Model Builder

- Integral tool in the SimSE process modeling approach
- Facilitates high-level construction of simulation models
- Hides the underlying process modeling language from the modeler – no programming required
- Current models:
  - waterfall (complete)
  - incremental (complete)
  - XP (95% complete)
  - RUP (95% complete)
  - inspection (complete)

(demo)

Different Kinds of Use

- Beginning students
  - Learning through illustration
  - Learning through decision-making
- Advanced students
  - Learning by building

Experiment Setup

- 29 test subjects played the game
  - each had taken at least an introductory software engineering course
- In 3 hours, subjects
  - learned the game
  - played it once or twice
  - filled out a questionnaire on the game

Numerical Results

Question	*1*	*2*	*3*	*4*	*5*	Avg
How enjoyable? (1=least enjoyable, 5=most enjoyable)	1	1	12	10	3	3.5
How difficult/easy? (1=most difficult, 5=easiest)	0	7	9	11	1	3.2
Reinforces material taught in class? (1=not at all, 5=definitely)	0	2	9	8	6	3.7
Teaches new process knowledge? (1=not at all, 5=definitely)	3	14	6	4	1	2.5
Teaches SE process in general? (1=not at all, 5=very much so)	0	2	12	10	4	3.6
Incorporate into SE course? (1=not at all, 5=very much so)	0	3	12	6	5	3.5
As an optional part? (1=not at all, 5=very much so)	0	6	8	7	6	3.4
As a mandatory part? (1=not at all, 5=very much so)	1	6	9	8	4	3.3

Written Feedback - Positive

- *"It does a good job of reinforcing the process in a very fun way!"*
- *"[It] makes [software engineering] seem more real and makes it more enjoyable."*
- *"[My favorite aspect of the game was] managing a team of employees. It is cool to see how they react to certain environments, and see how the project develops according to the selection of employees for different jobs."*
- *"52 teaches the intellectual level, overall view; the game illustrates this by feel and trial/error."*
- *"[Having to deal with] pay, energy, and mood introduces more complex, real-life issues present in a workspace."*

Feedback - Negative

- Game play a little bit idiosyncratic
- Additional approaches beyond waterfall
- Relationship of final score to actual play is unclear

A Repeat Experiment

- Use at the University of Applied Science, Konstanz, Germany
  - project engineering course
  - build a SimSE model of a hardware/software co-development process
- Very positive feedback
  - students found the experience very motivating
  - form of active knowledge acquisition
  - enforced explicitness and precision

Current Status

- Completed first version of simulation environment
  - 50,000 lines of Java code
- Next
  - build more models
  - further experimentation
  - class use
    - currently in use in a course at UC Irvine
  - explanatory tool
- The game and various models are downloadable

EASEL

- A novel software design environment that
  - supports master-apprentice learning
  - moves away from a focus on correct notation
  - allows incremental exploration
  - separates concerns
  - and eventually...
    - tolerates inconsistency
    - includes instant analyses / simulations / tradeoffs

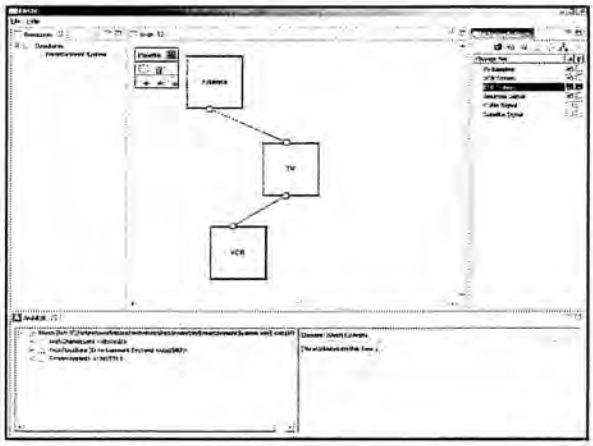
(note: this is early work in progress)

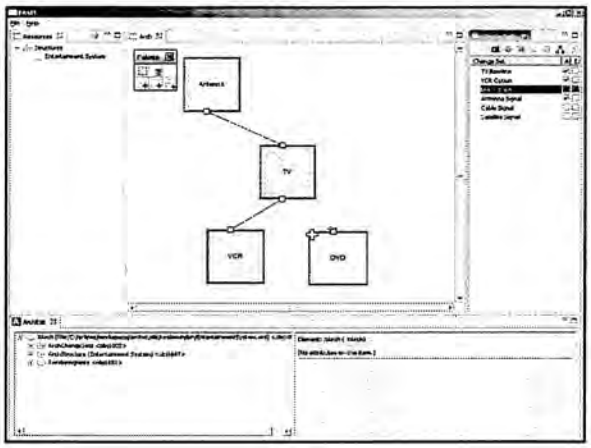
Design Principles of Layers and Groups

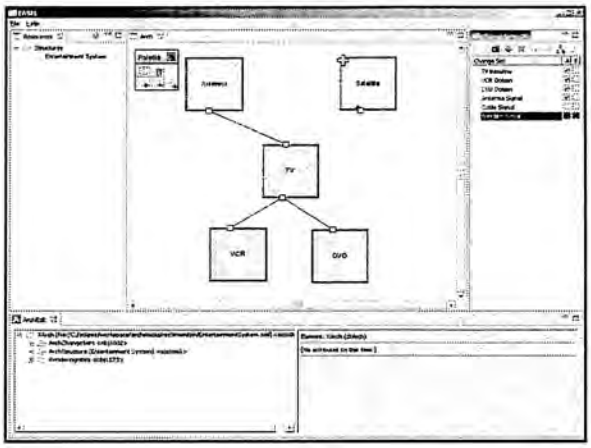
- Layers and groups are present in virtually any kind of design environment
  - graphical design
  - mechanical design
  - CAD/CAM design
  - architecture design
  - ...
- Layers and groups are not present in software design environments

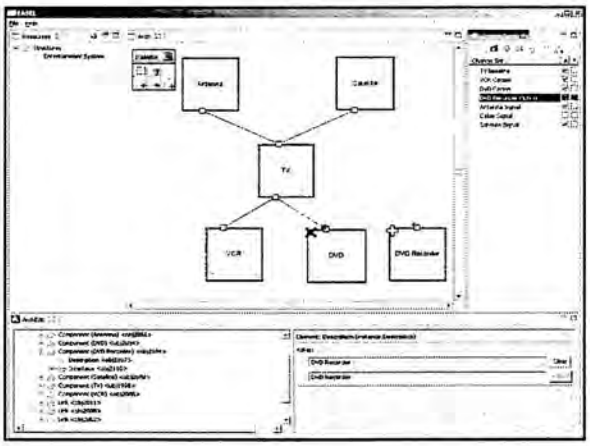
Layers and Groups in Software Design

- Layers can represent different "things"
  - change increments
  - alternative explorations
  - existing designs
  - patterns
  - refinements
- Groups can represent different "things"
  - concerns
  - views









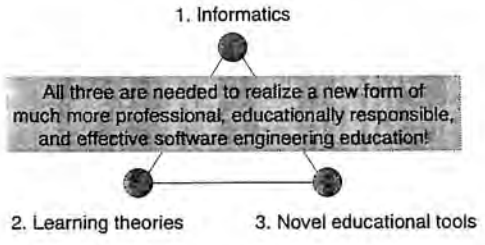
Educational Potential

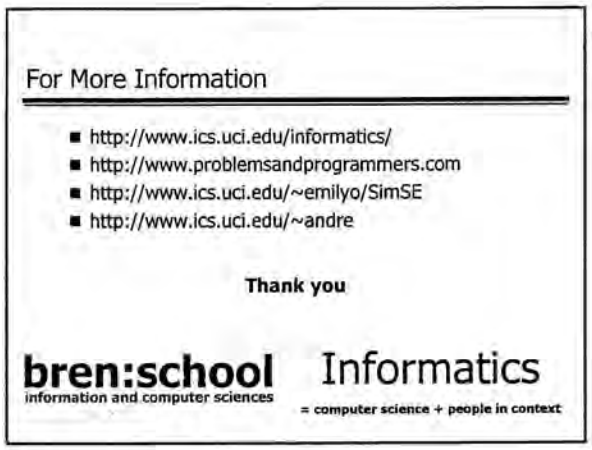
- Master-apprentice learning
  - "Change this design to do something slightly different"
  - "Create a new design, but it should follow the style of this other design (or set of patterns)"
  - "Examine the structure of this design and explain why it is good"
  - ...
- Instant feedback
  - simulations and analyses
- Creative exploration
  - ease of trying out new things without breaking the existing design
  - powerful undo

Observations

- Process and design are two areas ripe for improvement
  - very difficult to teach
  - require extensive practice
  - need for building an "intuition"
  - no tool support currently available specifically for education
- I have shown some examples of tools that can be used complementary to existing class delivery methods
  - novel focus
  - novel interaction mechanisms

Bringing It All Together




For More Information
http://www.ics.uci.edu/informatics/
http://www.problemsandprogrammers.com
http://www.ics.uci.edu/~emilyo/SimSE
http://www.ics.uci.edu/~andre
Thank you
bren:school
information and computer sciences
Informatics
= computer science + people in context

特別講演　「富山売薬と情報」

米原 寛（立山博物館 館長）

富山を語るキーワードとしては，売薬・北前船・立山信仰などがあるが、ソフトウェアとの関連を考えて，「富山売薬と情報」に的を絞ってお話しいただく．実際，情報とその流通いう視点は富山売薬を語る上できわめて重要である．

Ⅰ　富山売薬の特色

□形態的特色

1．組織としての売薬業・・・・仲間組を形成

売薬人の個別的な商売ではない。

営業権は「株立て」である。個人的な新懸けは禁止。重置は禁止。

「組」に集約され、個別商売は禁止される。

「組」：明和・安永期・・・１８組　　化政期：２０組　　天保期：２２組

仲間規約遵守のもとに営業可能。仲間組による［自主統制、自主管理］

2．半官（藩の役人）半民（町人・売薬人）による第三セクター経営

反魂丹役所（文化１３年設置）を基軸とした経営

・藩の役人は、藩の機関として売薬人に対する保護・統制を行い、

・有力売薬人は、株仲間に対する諸役金の徴収、株の掌握、諸通達の交付、諸出願の受付・処理、仕入金・雑用の貸付などの実務を行った。

3．富山藩の基幹産業

冥加金・運上金・営業税等の藩に納める諸税の総額は、２０００両～３０００両

□広域的商圏を形成・・・「差留」等の措置を克服

（1）経営の巧みさにより

（2）領国経済を熟知

（3）情報の活用

□経営の特色

1．経営の巧みさ

（1）薬効の高さ：原材料に高貴薬を使用。

原材料の吟味厳重　　薬種改座・薬種会所の設置、薬種業者の株極

（2）綿密な市場調査

（3）信用と信頼：売薬人に対して、薬に対して

□顧客との関係は、個別的、継（永）続的

□「先用後利」の理念に基づく利（使）用者側に立った商売

□顧客（得意先）のニーズを優先
- 薬のニーズ（配置の先見性）
- 文化のニーズ

（4）土産・進物等による気遣い　　売薬版画（絵紙）、小間物、昆布等

2．財務会計の巧みさ

（1）「懸場帳」のユニークさ

□記載内容：得意先の住所・氏名、配置日時、配置薬種名・数量・価格、前の配置における消費高、集金高、掛取高、その他特記事項

（2）「懸場帳」１冊の内容は、１日で回れる範囲（地理的）の顧客数

「懸場帳」１冊が動産、売買可能、売薬人１株の客体

（3）集金の輸送と換算の独特のシステム（両替機関の設置）

（4）反魂丹役所における会計処理は、複利計算、貨幣換算処理

Ⅱ　売薬人にとっての「情 報」

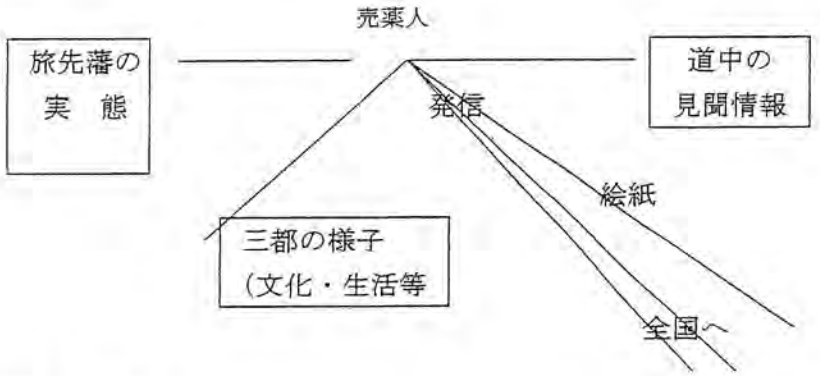
□情　報　　1．直接的情報：生起した事実に関する情報

2．間接的情報：見聞等による主観的情報

□売薬人にとっての情報

・旅先藩の気候・風土・人情・習俗・言語、、道路（街道）事情、政治・経済事情、文化事情

・旅行に伴う様々な体験・見聞による情報



□情報処理機関

反魂丹役所、組（年行司）

ソフトウェア技術者協会
〒160-0004 東京都新宿区四谷3-12 丸正ビル5F
Tel:03-3356-1077　Fax:03-3356-1072
E-mail:sea@sea.or.jp
URL:http://www.sea.jp/