EECS 4314

Advanced Software Engineering

Topic 07:

Reflexion Models and Source Sticky Notes

Zhen Ming (Jack) Jiang

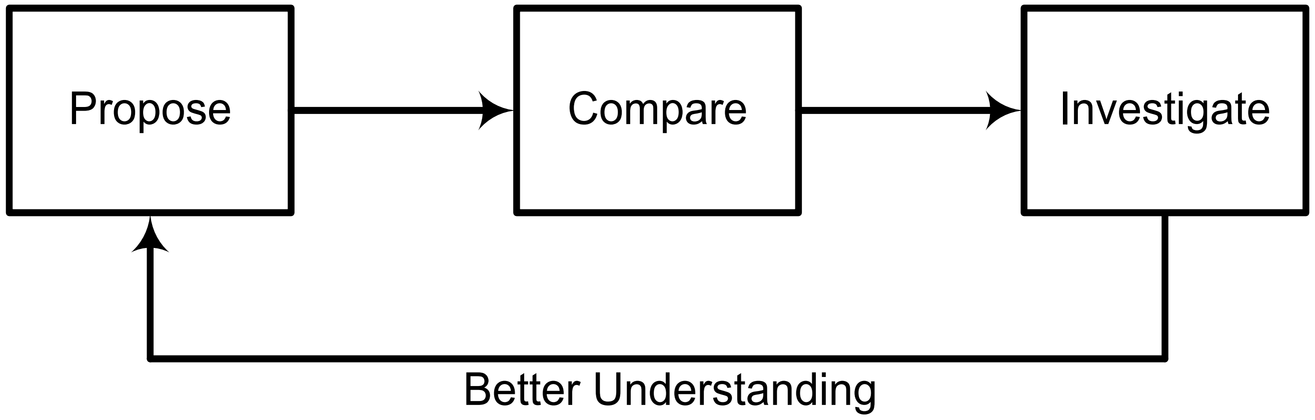
Relevant Readings

- Gail C. Murphy, David Notkin, and Kevin J. Sullivan. FSE 1995.
- Ahmed E. Hassan and Richard C. Holt. Using Development History Sticky Notes to Understand Software Architecture. IWPC 2004.

Introduction

- Software understanding tasks represent 50-90% of maintenance efforts
- Good documentation can help, but rarely available
- Some developers resort to code browsing, but that is limited and does not scale
- Propose to speedup understanding using knowledge from **historical modification records**

Architecture Understanding Process



- *Propose* a conceptual architecture
- *Compare* the conceptual with the concrete architecture
- *Investigate* gaps

Propose - Conceptual Architecture

- Developers propose a conceptual architecture based on:
  - Reference architecture
  - System documentation
  - Developer experience with similar systems
  - Talking to senior developers and domain experts

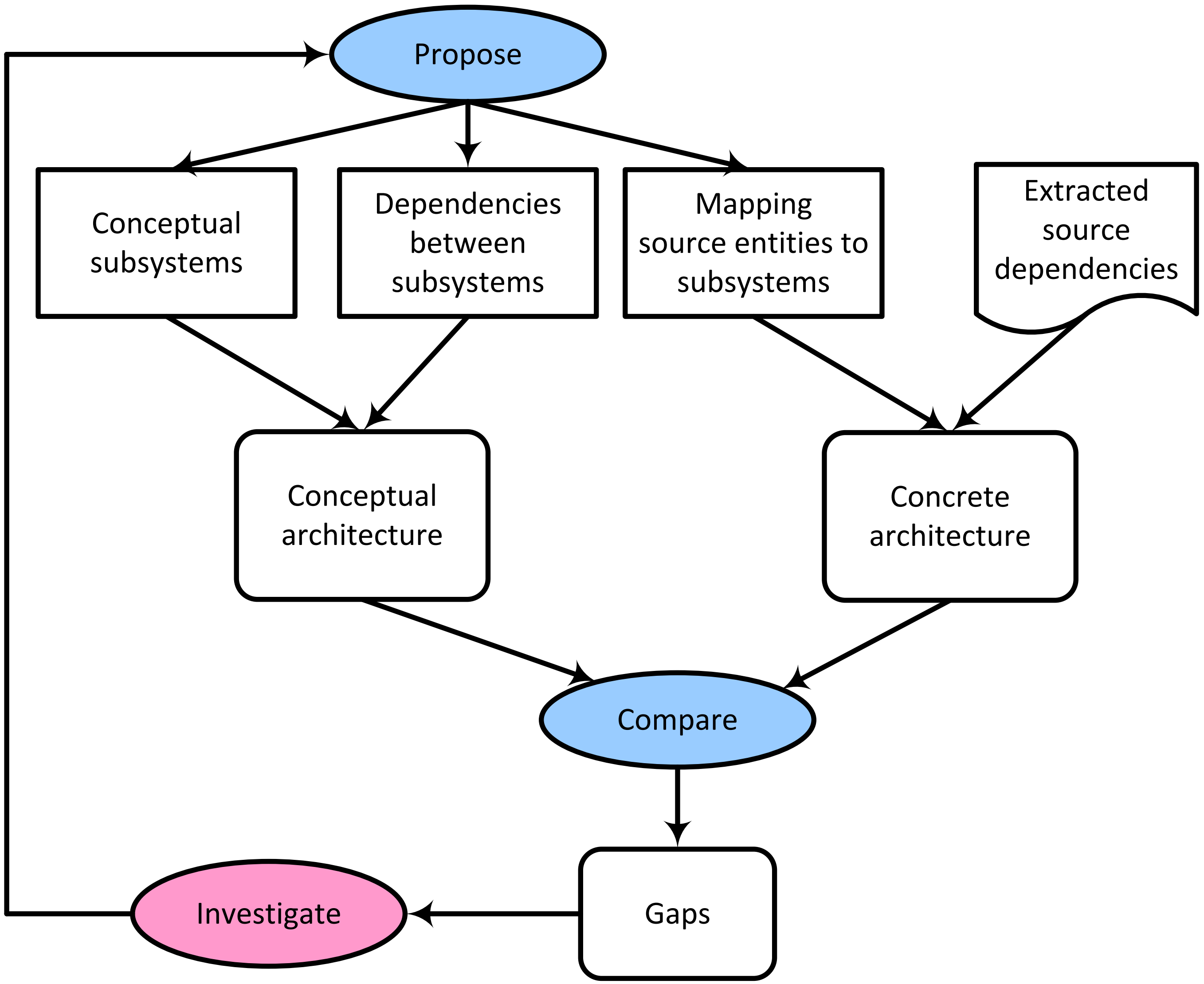
Mismatch between the Conceptual and Concrete Architecture

- However, in reality the concrete architecture is (almost) always different
- Need to not only discover the differences, but also uncover the rationale

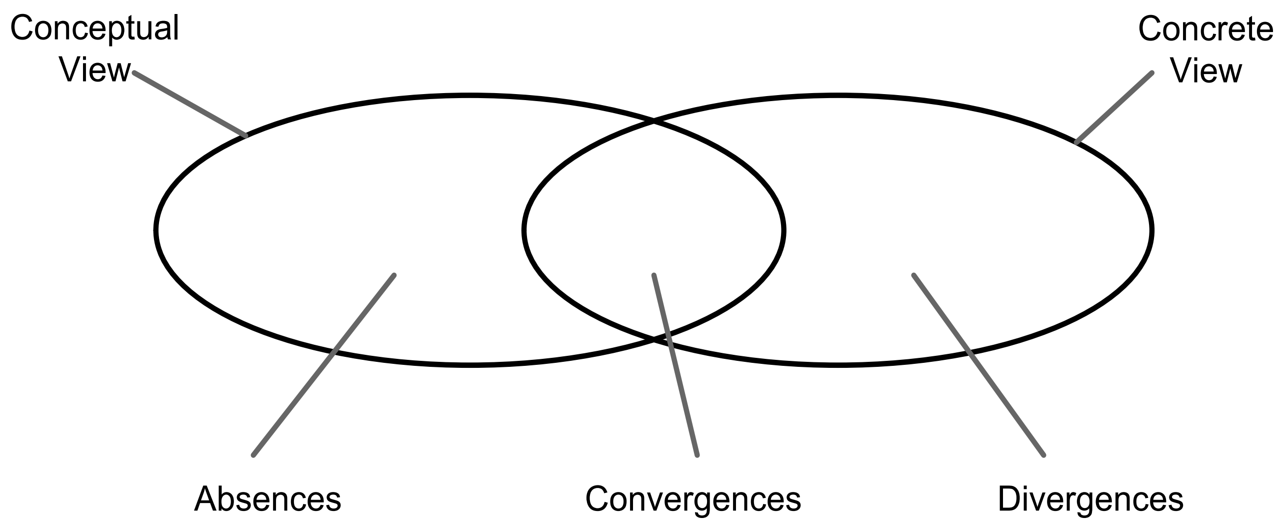
Uncovering the Rationale for the Differences

- Uncovering the rationale is challenging
  - A senior developer
    - may be too busy
    - may not recall the rationale for such dependency
    - may no longer work on the software system
  - The software
    - may have been bought from another company
    - may have its maintenance out-sourced
- Developers must spend hours/days to uncover the rationale. The rationale may be:
  - Justified due to, e.g., optimizations or code reuse; or
  - Not justified due to, e.g., developer ignorance or pressure to market.

Software Reflexion Framework


Propose
Conceptual subsystems
Dependencies between subsystems
Mapping source entities to subsystems
Extracted source dependencies
Conceptual architecture
Concrete architecture
Compare
Gaps
Investigate

Investigating Gaps



- ***Absences:*** rarely occur in large systems
- ***Convergences:*** usually not a concern
- ***Divergences:*** must investigate dependencies

Source Sticky Notes

- Attach change details to dependencies between software entities
- Provide insight to developers about reasons for that dependencies

4 “W”s when Investigating Dependencies

- Which
- Who
- When
- Why

Which

- Which concrete source code entities are responsible for an unexpected dependency?

Who?

- Who introduced an unexpected dependency or removed a missing dependency?

- A gap due to a change made by
  - a **novice developer** may suggest that the developer is at fault and the change must be fixed
  - a **senior developer** with a well established record for producing high quality code may suggest that the change is correct

When?

- When was the unexpected dependency added or the missing dependency removed?
  - Is it a fix to a critical bug under a tight release schedule?
    - E.g., a few days/hours before a release
  - Or is it a justified dependency that we should expect?

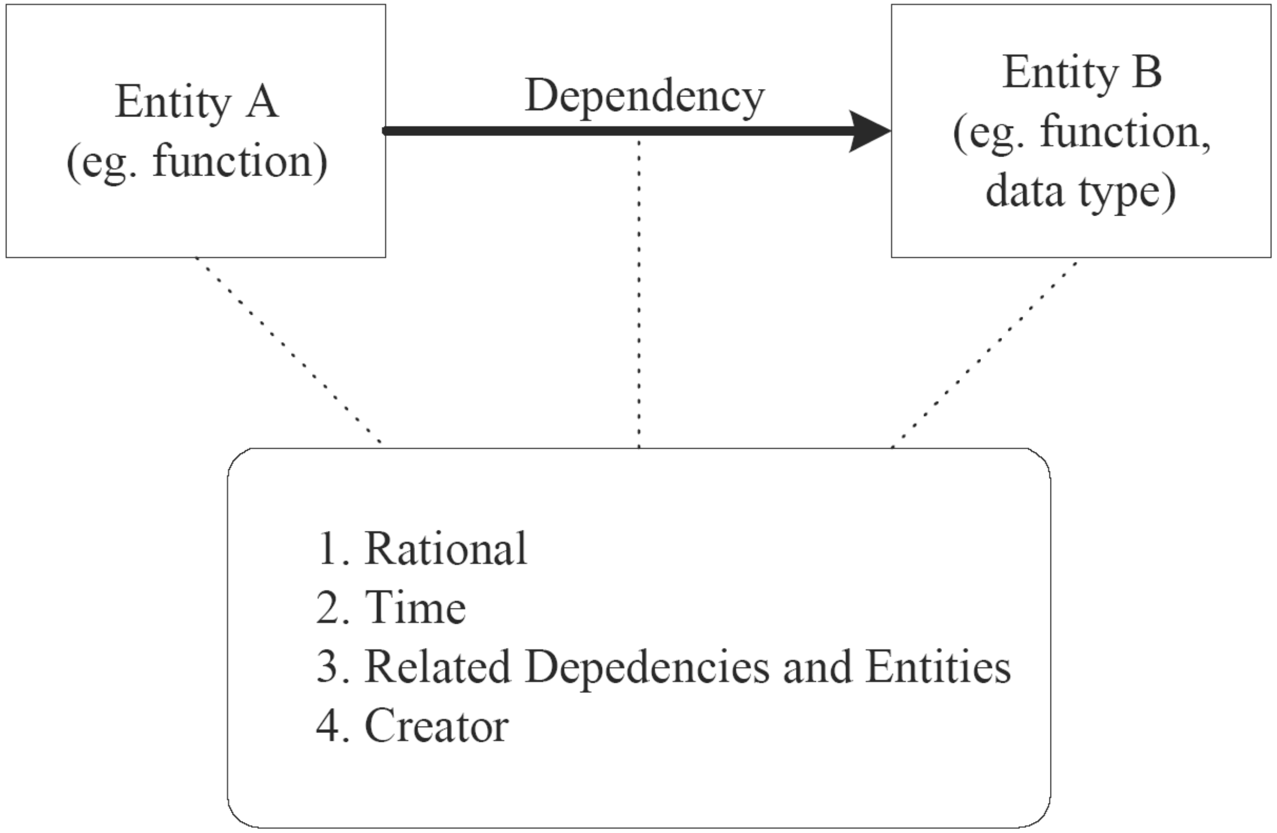
Why?

- Why was this unexpected dependency added or why was an expected dependency missing?
- Knowledge of the rationales is key in explaining the gaps

Dependency Investigation Questions (W4 Approach)

- ***Which*** low level code entity is responsible for the dependency?
  - Network (*SendData*) $\rightarrow$ Scheduler (*PrintToLog*)
- ***Who*** added/removed the dependency?
  - Junior vs. senior/experienced developer
- ***When*** was the dependency modified?
  - Late night / Just before release
- ***Why*** was the dependency added/removed?
  - The rationale!

Source *StickyNotes*



- We are interested in
  - Current and past dependencies

Source *StickyNotes*

- Static dependencies give only a current static view of the system – not enough detail!
- Need to extend static dependencies, but how?

Extending Code Dependencies

- Ask developers to fill StickyNotes for each change
  - Too time consuming and cumbersome
- Use software repositories to build these notes automatically
  - Historical information may be hard to process

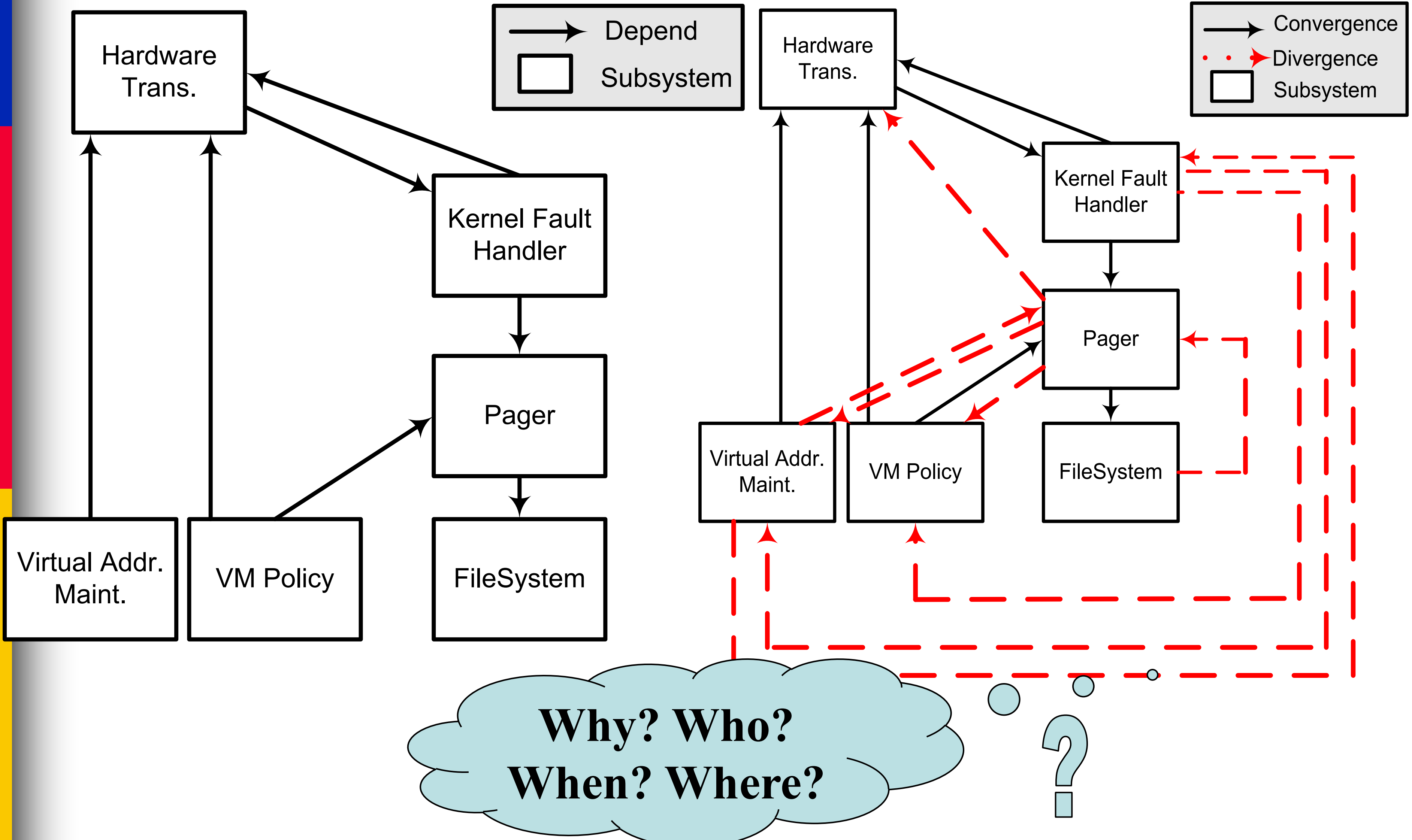
StickyNotes Recovery

- Map code changes to entities and dependencies instead of lines
- Two pass analysis of the source control repository data, to recover:
  - Record all entities defined throughout the lifetime of a project
  - Record all dependencies between these entities and attach source control meta-data

Case Study – NetBSD

- Large long lived system with hundreds of developers
- Case study used to demonstrate usefulness of the reflexion model:
  - Reuse prior results! ☺
  - Focus on investigating *gaps* to show the strength of the approach of the historical sticky notes

NetBSD's Virtual Memory Component Conceptual and Reflexion Architecture



Unexpected Dependencies

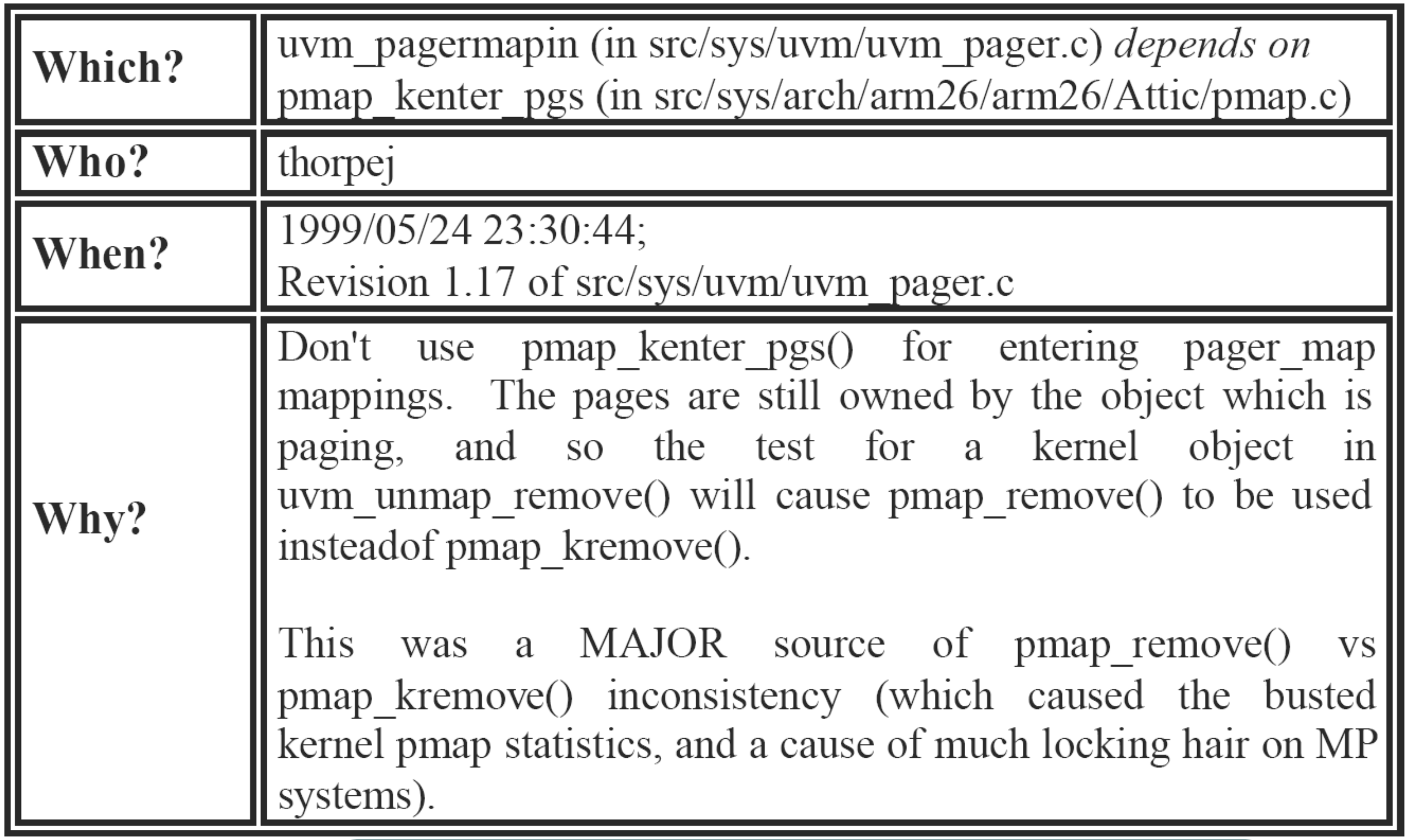
- Eight unexpected dependencies
- All except two dependencies existed since day one:
  - Virtual Address Maintenance → Pager

Which?	vm_map_entry_create (in src/sys/vm/Attic/vm_map.c) *depends on* pager_map (in /src/sys/uvm/uvm_pager.c)
Who?	cgd
When?	1993/04/09 15:54:59 Revision 1.2 of src/sys/vm/Attic/vm_map.c
Why?	from sean eric fagan: it seems to keep the vm system from deadlocking the system when it runs out of swap + physical memory. prevents the system from giving the last page(s) to anything but the referenced "processes" (especially important is the pager process, which should never have to wait for a free page).

Dependency added to avoid deadlocking under special circumstances

Unexpected Dependencies

- Pager → Hardware Translations

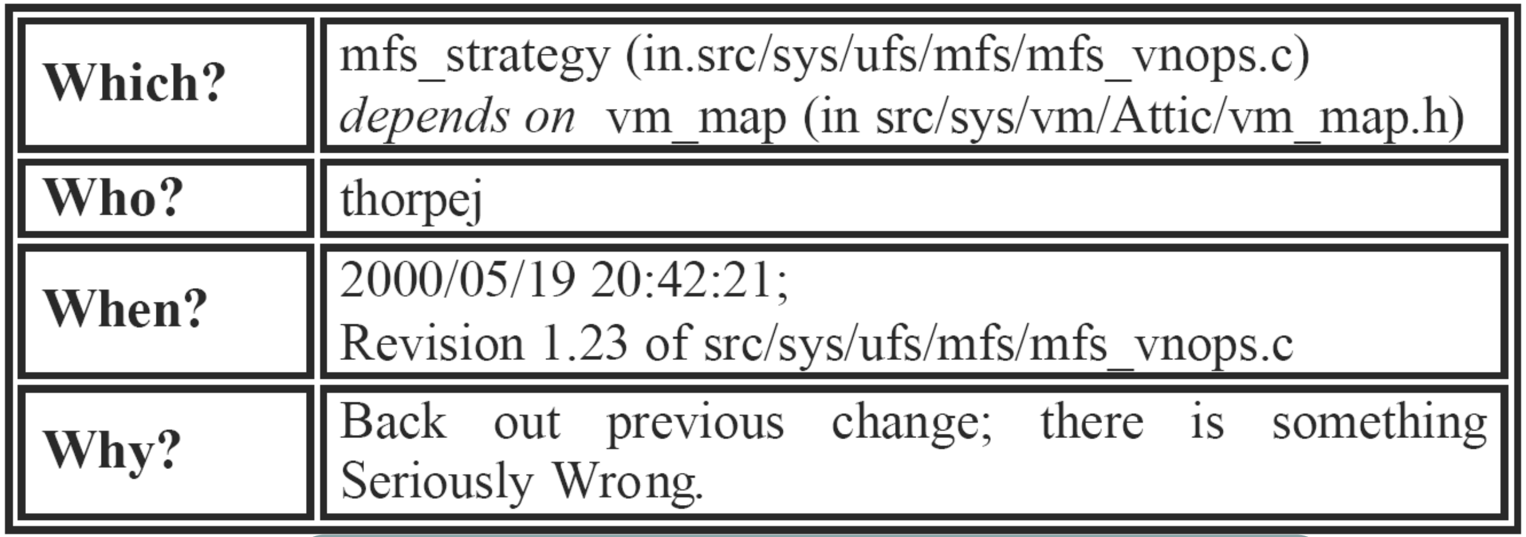


Which?	uvm_pagermapin (in src/sys/uvm/uvm_pager.c) *depends on* pmap_kenter_pgs (in src/sys/arch/arm26/arm26/Attic/pmap.c)
Who?	thorpej
When?	1999/05/24 23:30:44; Revision 1.17 of src/sys/uvm/uvm_pager.c
Why?	Don't use pmap_kenter_pgs() for entering pager_map mappings. The pages are still owned by the object which is paging, and so the test for a kernel object in uvm_unmap_remove() will cause pmap_remove() to be used insteadof pmap_kremove(). This was a MAJOR source of pmap_remove() vs pmap_kremove() inconsistency (which caused the busted kernel pmap statistics, and a cause of much locking hair on MP systems).

Dependency added to fix a bug on multiple process systems

Unexpected Dependencies which existed in the past

- Two unexpected dependencies that were removed in the past:
  - Hardware Translation ➞ VM Policy
  - File System ➞ Virtual Address Maintenance



Which?	mfs_strategy (in.src/sys/ufs/mfs/mfs_vnops.c) *depends on* vm_map (in src/sys/vm/Attic/vm_map.h)
Who?	thorpej
When?	2000/05/19 20:42:21; Revision 1.23 of src/sys/ufs/mfs/mfs_vnops.c
Why?	Back out previous change; there is something Seriously Wrong.

Dependency removed to fix a previous incorrect change

StickyNotes Usage Patterns

- ***First note*** to understand the reason for unexpected dependencies
- ***Last note*** to study missing dependencies
- ***All notes*** when first and last notes do not have enough information to assist in understanding

Limitations

- Quality of comments and text entered by developers in the past
- In many open source projects, code revision comments are used for:
  - Communicating new features
  - Narrating the progress of a project

Summary

- Development history can help understand the current structure of a software system
- Traditional dependency graphs and program understanding models usually do not use historical information
- Proposed *StickyNotes* and presented a case study to show the strength of the approach