Software Engineering

Vineet Seth

7th Feburary 2006

$$\sum^{SRS+Design+HLD/LLD+Coding+testing+Acceptance} \simeq \text{Product}$$

- Main Product $\simeq$ SRS

$$\sum^{SRS+Design+HLD/LLD+Coding+testing+Acceptance} \simeq \text{Product}$$

- Main Product $\simeq$ SRS
- Byproduct : User Manual, Install Manual, Packaging

$$\sum^{SRS+Design+HLD/LLD+Coding+testing+Acceptance} \simeq \text{Product}$$

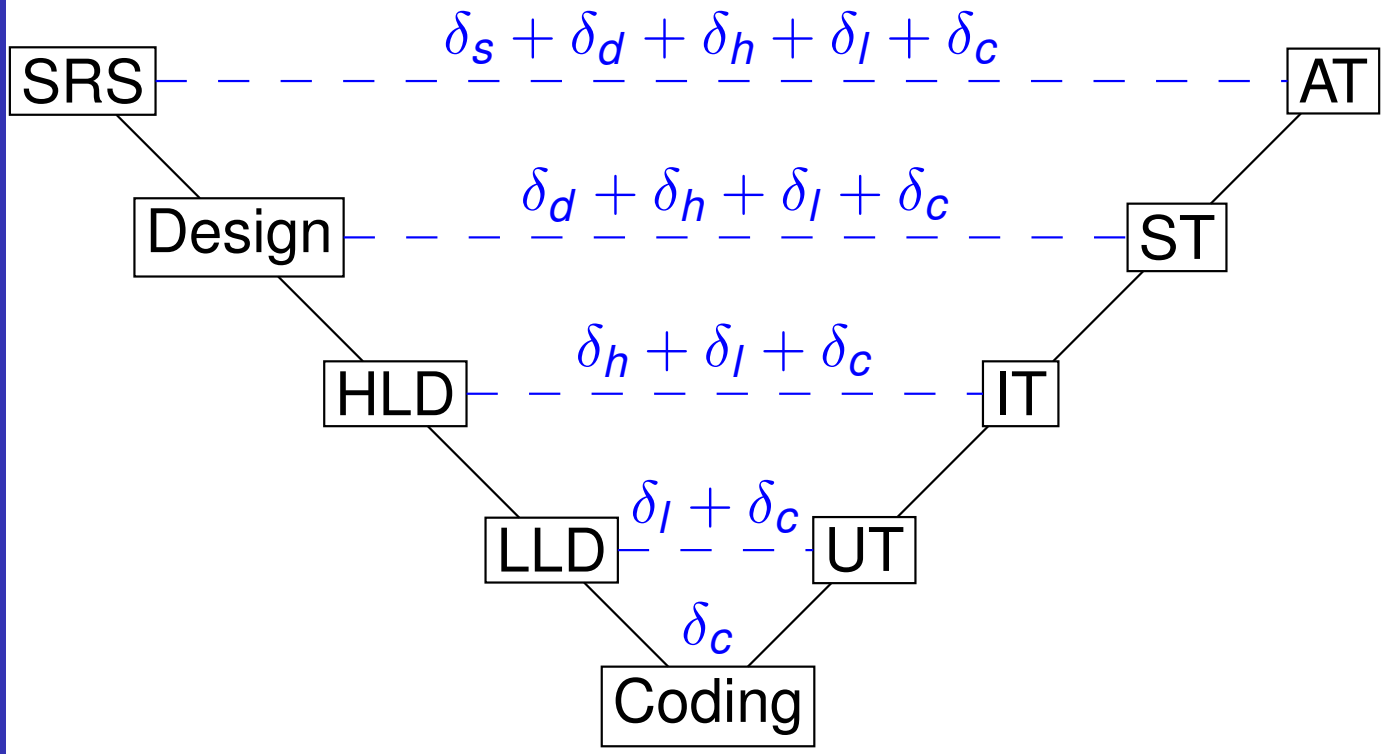
- Main Product $\simeq$ SRS
- Byproduct : User Manual, Install Manual, Packaging
- Secondary Product (Waste Products) : Produced for transaction between the phases

$$\sum^{SRS+Design+HLD/LLD+Coding+testing+Acceptance} \simeq \text{Product}$$

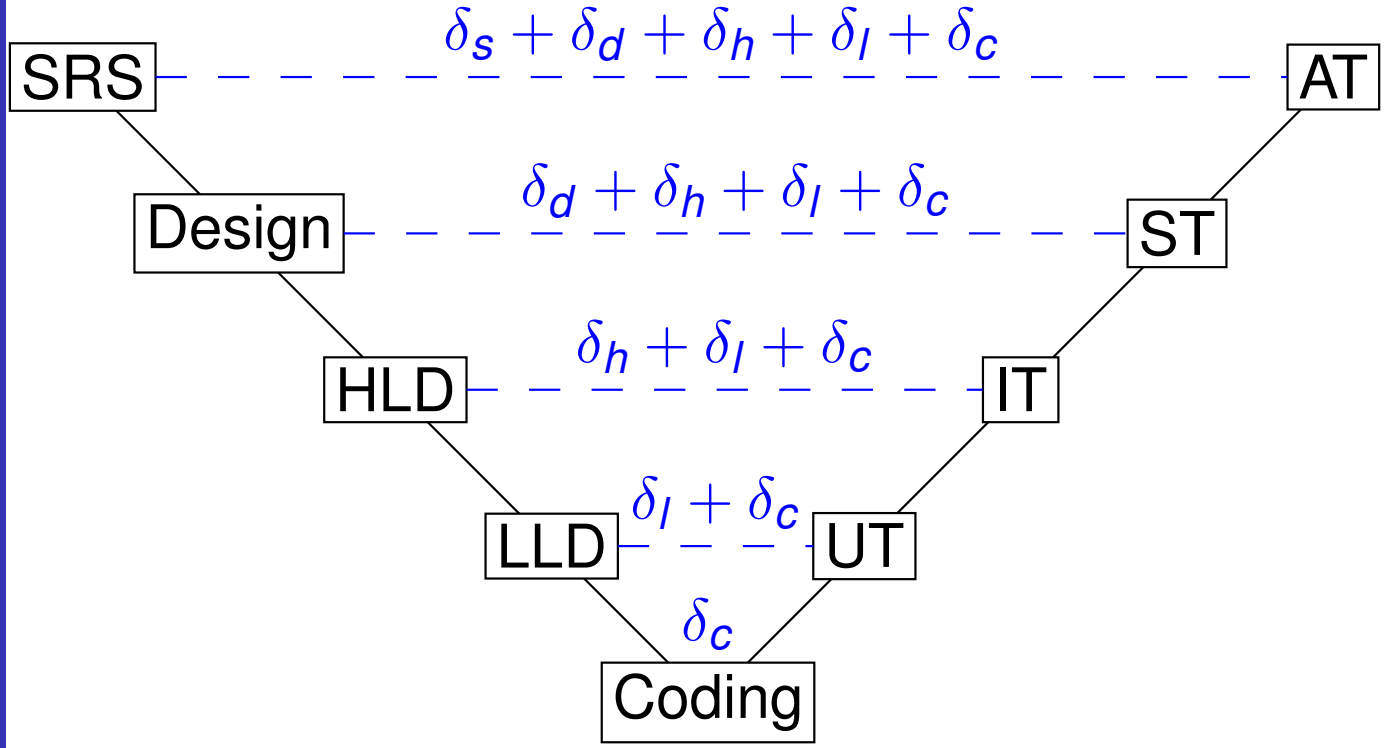
- Main Product $\simeq$ SRS
- Byproduct : User Manual, Install Manual, Packaging
- Secondary Product (Waste Products) : Produced for transaction between the phases

Assumptions

- Development is a sequential model
- Every phase is connected to each other
- Flaw at any stage means updating to referring stages



$\delta_x \longleftarrow$ *is the change Factor*



$\delta_x \longleftarrow$ *is the change Factor*

- Based on assumption that ϕ has perfect output and is complete
- Failure at any stage, means referring back to previous ϕ's
- Bug detection means cycling through the ϕ's(δ factor)

Problem with V model

- The output of each ϕ has to be in sync with previous ϕ's
- Detection of a problem at any stage will mean updating previous stages

Problem with V model

- The output of each ϕ has to be in sync with previous ϕ's
- Detection of a problem at any stage will mean updating previous stages
- Laziness will soon cause the gaps between the referring stages
- Frustration of developer to follow standards will create more gaps

Problem with V model

- The output of each ϕ has to be in sync with previous ϕ's
- Detection of a problem at any stage will mean updating previous stages
- Laziness will soon cause the gaps between the referring stages
- Frustration of developer to follow standards will create more gaps
- Coding, design and Test Documents are not in Sync by the time testing cycle is through

Problem with V model

- The output of each ϕ has to be in sync with previous ϕ's
- Detection of a problem at any stage will mean updating previous stages
- Laziness will soon cause the gaps between the referring stages
- Frustration of developer to follow standards will create more gaps
- Coding, design and Test Documents are not in Sync by the time testing cycle is through

Eventually the Quality suffers

- Frustration of newer staff and Maintenance problem

- Reduce the δ accumulation factor
- Automate the output of previous stages as much as possible
- Even if partial automation is done, δ factor gets reduced

- Reduce the δ accumulation factor
- Automate the output of previous stages as much as possible
- Even if partial automation is done, δ factor gets reduced

Why not do it?

- We are still not convinced with a different approach
- The team members understand the adopted practices
- Client dictates the way we proceed
- Our SQA is happy with the way we proceed
- Management insists that we follow older approach

- Reduce the δ accumulation factor
- Automate the output of previous stages as much as possible
- Even if partial automation is done, δ factor gets reduced

Why not do it?

- We are still not convinced with a different approach
- The team members understand the adopted practices
- Client dictates the way we proceed
- Our SQA is happy with the way we proceed
- Management insists that we follow older approach

Don't change the model,Just automate it

- Highly skilled people of 60 or so
- The product is well designed and Documented

College student with half the skill provides a solution

- Highly skilled people of 60 or so
- The product is well designed and Documented

College student with half the skill provides a solution

To be more competitive

- Product has to have more features(Cost for Money)
- Questions on the product sanity and performance begins to arise
- Marketing team works harder to sell
- Designer work overtime to make the product look good
- Enforcement of heavy deadline

- Developer works harder
- Problem in updating the documents and keeping its sanity
- Coping up with the frustrations of overworked developer, Marketing people.
- Answering to the management about missing deadline
- Management Rationalize their product as superior quality, with full support etc...

- Developer works harder
- Problem in updating the documents and keeping its sanity
- Coping up with the frustrations of overworked developer, Marketing people.
- Answering to the management about missing deadline
- Management Rationalize their product as superior quality, with full support etc...

How can this college student cause this much confusion?

- Self Motivated (No money is being charged)
- Well studied and build his software based on ideology from some Journal
- Gets good backing from the University's

Tasks of Independent Developer

- Maintaining at-least 3 New groups List or more: developer/user/announcement
- Updating the code and its patches as discussed in the developer list
- Updating FAQ, User Manual, Design docs etc. . .
- Monitoring the security of the site
- Plus a part time job, studies, attending conferences etc. . .
- and more. . .

Tasks of Independent Developer

- Maintaining at-least 3 New groups List or more: developer/user/announcement
- Updating the code and its patches as discussed in the developer list
- Updating FAQ, User Manual, Design docs etc. . .
- Monitoring the security of the site
- Plus a part time job, studies, attending conferences etc. . .
- and more. . .

Output is the same, except that its more effective. . .

Known Facts

- We anyway pick up 70%-80% of code from Web, and sometimes whole of it.
- They have much better algorithm
- The have a larger user base
- Their code runs

Known Facts

- We anyway pick up 70%-80% of code from Web, and sometimes whole of it.
- They have much better algorithm
- The have a larger user base
- Their code runs

Adopting

- Why not adopt their design methods also?
- We have nothing to lose
- We get the advantage to understanding their model and code better?

1. Become impatient
   If you are repeating the same task over and over again you are doing it the wrong way

1. Become impatient
   If you are repeating the same task over and over again you are doing it the wrong way
2. Become Lazy
   Every key you press is waste of your energy.
   Question yourself whether you have to retype it

1. Become impatient
   If you are repeating the same task over and over again you are doing it the wrong way
2. Become Lazy
   Every key you press is waste of your energy.
   Question yourself whether you have to retype it
3. Get into the practice of older methods of using Pen and Paper
   Writing of code should be done on paper, then analyzed and rethought, redone again until satisfied.
   Finally typed onto the computer

1. Become impatient
   If you are repeating the same task over and over again you are doing it the wrong way
2. Become Lazy
   Every key you press is waste of your energy.
   Question yourself whether you have to retype it
3. Get into the practice of older methods of using Pen and Paper
   Writing of code should be done on paper, then analyzed and rethought, redone again until satisfied.
   Finally typed onto the computer
4. Give yourself time and decide where to put in efforts.
   Developing skills and gaining knowledge is more important

Consider every piece of software you develop has already been done

- You got no new ideas
- You don't understand Open Source model

Result of Understanding of Open-Source Model

- Understanding of Open Source model
- Modification of Open Sources
- Doing things the way Open source does it, which is the faster way
- Meeting the deadlines

Consider every piece of software you develop has already been done

- You got no new ideas
- You don't understand Open Source model

Designing is one of the the most difficult topics. It dictates the outcome of the project. If a design is assumed to be correct based on the Experience of the Designer, then it is a major downfall. You can assume then and there, that the execution will have problems

Facts

- Designing in not about producing the document, but a proof that the design will work.
- Design can only be of 2 forms

Designing is one of the the most difficult topics. It dictates the outcome of the project. If a design is assumed to be correct based on the Experience of the Designer, then it is a major downfall. You can assume then and there, that the execution will have problems

Facts

- Designing in not about producing the document, but a proof that the design will work.
- Design can only be of 2 forms
  - Based on a working prototype.
  - Based on a Mathematical proof

Designing is one of the the most difficult topics. It dictates the outcome of the project. If a design is assumed to be correct based on the Experience of the Designer, then it is a major downfall. You can assume then and there, that the execution will have problems

Facts

- Designing in not about producing the document, but a proof that the design will work.
- Design can only be of 2 forms
  - Based on a working prototype.
  - Based on a Mathematical proof

Any other design document is not a design, e.g for reimplementation of a RFC, RFC is the design whose proof of concept has already been provided by the RFC author.

- Know the subject throughly
- What are the open Sources available
- From the SRS gather what is the primary target
- Assume that the SRS is going to change and is incomplete. (In most cases it is anyway)
- Get the framework ready (Secondary Targets)

  `This is the area where I will be concentrating my efforts`

Outcome from the Design Phase:

- Organization of the Sources
- User Manual and Install Manual framework
- All the additional libraries and packaging
- Test case plan infrastructure

- Knowing the language well does not mean that you know the features of the compiler
- Indentation is important
- Documentation of code is not important, Indentation is the best document.

- Knowing the language well does not mean that you know the features of the compiler
- Indentation is important
- Documentation of code is not important, Indentation is the best document.
  1. Only document /* TO-DO: */
  2. Documentation of code gets into the way of coding, also its not a virtue of a lazy person
  3. Document only obscure coding, that gets into the way of reading.

- Knowing the language well does not mean that you know the features of the compiler
- Indentation is important
- Documentation of code is not important, Indentation is the best document.
  1. Only document /* TO-DO: */
  2. Documentation of code gets into the way of coding, also its not a virtue of a lazy person
  3. Document only obscure coding, that gets into the way of reading.

Exceptions

User Manual and code Documentation can be generated from the code, which makes maintainability easy.

- Study/Understand a open source code, its coding conventions and techniques.
- Use Macros & Compiler features as much as possible.
  E.g : It is not enough to know K&R. Use must be able to use compiler features
- For portability try different environment
  e.g For Open-Source development, machine type is not an issue as they have Compile farms available
  So we should be better off then some open-source person using P-I machine with 64 MB Ram or less, being able to develop portable code across all the platforms.

- Know the scripting languages well
- Try to automate any thing and everything even if its manual e.g If manual testing is the only way then get as much automation done a possible.
- Test-suite should be ready to use.
- Know the class to people to report.

- Know the scripting languages well
- Try to automate any thing and everything even if its manual e.g If manual testing is the only way then get as much automation done a possible.
- Test-suite should be ready to use.
- Know the class to people to report.

Facts

- Testing is not QA activity.
- Its a Misconception that testing improves the Quality of the product.
- Testing is a validation process.
- Testing is the only way to increase your domain knowledge
- Testing provides insight into the compiler, tools etc. . .

- Everybody knows and accepts documentation is important and many of us have been a part of it.
- How many of us have produced Man pages, Info pages, HTML help, Windows help. . . ?
- How many of us have used Perl-doc, Java-doc, doxygen inside the code to produce documentation. . . ?
- How many of us have used LaTeX, Doc-Book to produce documentation. . . ?

- Everybody knows and accepts documentation is important and many of us have been a part of it.
- How many of us have produced Man pages, Info pages, HTML help, Windows help. . . ?
- How many of us have used Perl-doc, Java-doc, doxygen inside the code to produce documentation. . . ?
- How many of us have used LaTeX, Doc-Book to produce documentation. . . ?

We are used to producing WYSIWYG documentation based on M$ WORD but even M$ also does not provide help using Word

`The problem is it does not merge well with Software Development`

Why change it?

- Because there is no way to automate it...(in case a change is required)
- Because there are distributed as word doc, and cannot be rendered into other forms like pdf, HTML, info
- Because it distracts the authors into the looks of the document, rather than content. . .

Why change it?

- Because there is no way to automate it...(in case a change is required)
- Because there are distributed as word doc, and cannot be rendered into other forms like pdf, HTML, info
- Because it distracts the authors into the looks of the document, rather than content. . .

Notice how FAQ are maintained, you can get them either as PDF, PS, RTF, HTML

- Way it is done is either LaTeX, Doc-Book(SGML) <– 1 source multiple outputs.

- Way it is done is either LaTeX, Doc-Book(SGML) <– 1 source multiple outputs.
- LaTeX is used for Technical Documents e.g Journals, Proposals, Data Sheets, etc. . .
- Doc-Book is used for Publications e.g all the O'Reilly Publication are of Doc-Book type.

- Way it is done is either LaTeX, Doc-Book(SGML) <– 1 source multiple outputs.
- LaTeX is used for Technical Documents e.g Journals, Proposals, Data Sheets, etc. . .
- Doc-Book is used for Publications e.g all the O'Reilly Publication are of Doc-Book type.
- As for the HLD/LLD part, their a host of tools e.g dogygen to produce document from code.
  If properly configured with Graph-Wiz, it can produce graphical diagrams.
  Many of the libraries distributed, produce documents this way.
  It is easier to maintain code in-sync with documentation this way. . .

- Way it is done is either LaTeX, Doc-Book(SGML) <– 1 source multiple outputs.
- LaTeX is used for Technical Documents e.g Journals, Proposals, Data Sheets, etc. . .
- Doc-Book is used for Publications e.g all the O'Reilly Publication are of Doc-Book type.
- As for the HLD/LLD part, their a host of tools e.g dogygen to produce document from code.
  If properly configured with Graph-Wiz, it can produce graphical diagrams.
  Many of the libraries distributed, produce documents this way.
  It is easier to maintain code in-sync with documentation this way. . .
- A change in the way we use the documentation can speed up our work by 30%-40% at-least and that too will keep every thing in sync . . .

Make utility

This utility is perhaps the core to every development cycle, and should be freezed during the design phase itself.
You can use this utility to achieve full automation right from the producing
Design Documents $\rightarrow$ Coding $\rightarrow$ Testing $\rightarrow$ Results $\rightarrow$ User Manual creation $\rightarrow$ Packaging

Make utility

This utility is perhaps the core to every development cycle, and should be freezed during the design phase itself.
You can use this utility to achieve full automation right from the producing
Design Documents $\rightarrow$ Coding $\rightarrow$ Testing $\rightarrow$ Results $\rightarrow$ User Manual creation $\rightarrow$ Packaging

Accessories

There are accessories to make, which can be classified as `Autoconf, Automake` This is just a set of m4 macros which you can use to configure you code for portability.

- Perhaps the most open topic. . . , many different approaches
- Deja-GNU ← Provides a testing framework which is Posix 1003.3 compliant, and TeT compliant.
  Uses expect and tcl

- Perhaps the most open topic. . . , many different approaches
- Deja-GNU ← Provides a testing framework which is Posix 1003.3 compliant, and TeT compliant.
  Uses expect and tcl
  Advantage:
  - Integrates well with Automake utility
  - Can be used to test any testing from Batch, Interactive, GUI testing.
  - GCC used it, GDB uses it
  - Learning to use Deja-Gnu will give you insight into the working of GCC, GCJ, GDB. . . glibc uses Perl
  - Can be used to test remote configuration

Automation

is the key to meeting deadlines. . .

Automation

is the key to meeting deadlines. . .

1.

For project's

Get the secondary targets ready (framework), focus on the primary target later. . .

Automation

is the key to meeting deadlines. . .

1.

For project's

Get the secondary targets ready (framework), focus on the primary target later. . .

2.

For Open-Source

Understand the Secondary before the Primary target

3.